

UNIVERSIDAD AUTÓNOMA DE MADRID  
ESCUELA POLITÉCNICA SUPERIOR



Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

DETECCIÓN DE LESIONES CUTÁNEAS EN  
IMÁGENES BASADO EN REDES  
GENERATIVAS ADVERSARIAS

Nicolás Alexander Wolyniec Rojas  
Tutor: Juan Carlos San Miguel  
Ponente: Jesús Bescos

Julio 2020



# **DETECCIÓN DE LESIONES CUTÁNEAS EN IMÁGENES BASADO EN REDES GENERATIVAS ADVERSARIAS**

**Nicolás Alexander Wolyniec Rojas**  
**Tutor: Juan Carlos San Miguel**  
**Ponente: Jesús Bescos**



**Video Processing and Understanding Lab**  
**Dpto. Tecnología Electrónica y de las Comunicaciones**  
**Escuela Politécnica Superior**  
**Universidad Autónoma de Madrid**  
**Julio 2020**



# Resumen

El propósito de este Trabajo Fin de Grado es conocer y poner en práctica las redes generativas adversarias en un problema real. Para ello, primero se estudia el panorama actual en el que esta arquitectura está siendo utilizada. Posteriormente, se crea una red GAN (generative adversarial network) y se evalúan los resultados.

Tras profundizar en la parte teórica se pasa a la práctica utilizando Keras y Kaggle (plataforma online de data science) para crear la red generativa. El objetivo de esta red es generar nuevas imágenes y utilizarlas como data augmentation en un problema de clasificación. Este problema de clasificación es la ISIC-2019, una competición a nivel mundial en la que se buscan modelos capaces de clasificar entre ocho tipos de cáncer de piel.

El flujo de trabajo es el siguiente: 1) La red creada genera imágenes de un tipo de cáncer de piel en particular (Actinic Keratosis). 2) Éstas imágenes se añaden al conjunto de datos original y 3) se realiza una comparación para ver si el modelo clasificador mejora o no introduciendo nuevas imágenes creadas.

Los resultados obtenidos muestran que no se ha sido capaz de mejorar el rendimiento del modelo clasificador con las nuevas imágenes generadas por la red GAN. Esto se debe por un lado a que la red GAN no es capaz de aprender lo suficiente, debido a la escasez de datos, y por otro a la carencia de poder computacional.

Aún a pesar de no obtener los resultados esperados, se ha cumplido el objetivo: ahondar en el conocimiento de las redes generativas adversarias y utilizarlas en un problema de la vida real. También, tras realizar este Trabajo Fin de Grado brotan una serie de preguntas para un estudio a posteriori: ¿Se puede aplicar el data augmentation con la arquitectura GAN en la mayoría de los problemas? ¿Cómo se puede afrontar un conjunto de datos muy desbalanceado, es decir, donde algunas categorías tienen mucho peso mientras que otras muy poco? ¿Puede una red GAN ser pieza diferenciante para este tipo de problemas?

## Palabras clave

Red Generativa Adversaria, GAN, ISIC-2019, Data Augmentation



# Abstract

The purpose of this Bachelor thesis is to introduce adversarial generative networks. To do this, we first observe the current panorama in which this architecture is being used. Subsequently, a GAN (generative adversarial network) is created and the results are evaluated.

Once the theoretical part is seen, Keras and Kaggle (online data science platform) are used to create the generative network. The objective of this network is to generate new images and use them as data augmentation for a classification problem. This classification problem is ISIC-2019, a worldwide competition that seeks models capable of classifying between eight types of skin cancer.

The workflow is as follows: 1) The network created generates a type of skin cancer (Actinic Keratosis). 2) These images are added to the original dataset and 3) a comparison is made to see if the classification model improves or not by introducing new images created.

The results obtained show that it has not been able to improve the performance of the classification model with the new images generated by the GAN network. This is because the GAN network is not capable of learning enough, due to a lack of data, and also due to a lack of computational power.

Even though the expected results have not been obtained, the objective has been met: to enter adversarial generative networks and use them in a real-life problem. Also, after completing this Bachelor thesis, a series of questions arise for a subsequent study: Can data augmentation be applied with the GAN architecture in most problems? How can you deal with a data set that is very unbalanced, that is, where some categories have a lot of weight while others have very little? Can a GAN network make a difference for this type of problem?

## Keywords

Generative Adversarial Network, GAN, ISIC-2019, Data Augmentation





# Agradecimientos

Quiero agradecer el tiempo y la pasión dedicada, tanto en clase como en este trabajo, por mi profesor Juan Carlos San Miguel. Ha sido un placer aprender y adentrarme en este mundo guiado por él.

También quiero reconocer el esfuerzo de mi mujer Carmen por haber estado ahí cuando lo necesitaba y haber sido un pilar durante estos cuatro años.



# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	1
1.2. Objetivos . . . . .	1
1.3. Organización de la memoria . . . . .	2
<b>2. Estado del arte</b>	<b>3</b>
2.1. Introducción . . . . .	3
2.2. Redes Neuronales Convolucionales (CNN) . . . . .	3
2.2.1. Convolución . . . . .	4
2.2.2. Filtrado no lineal . . . . .	4
2.2.3. Redimensionado espacial . . . . .	5
2.3. Redes Generativas Adversarias (GAN) . . . . .	7
<b>3. Competición ISIC</b>	<b>11</b>
3.1. Introducción . . . . .	11
3.2. Tabla comparativa . . . . .	11
3.3. Análisis de artículos que SÍ utilizaron GAN . . . . .	13
3.3.1. Towards Automated Melanoma Detection with Deep Learning: Data Purification and Augmentation . . . . .	14
3.3.2. Data Augmentation for Skin Lesion using Self-Attention based Progressive Generative Adversarial Network . . . . .	17
3.3.3. Skin Lesion Analysis Towards Melanoma Detection Using Gene- rative Adversarial Network . . . . .	18
3.4. ¿Por qué utilizar GAN en la detección de cáncer de piel? . . . . .	19
<b>4. Red Generativa Adversaria</b>	<b>21</b>
4.1. Introducción . . . . .	21
4.2. Definición . . . . .	21
4.3. Red Generativa . . . . .	22
4.4. Red Discriminativa . . . . .	24
4.5. Hiper Parámetros . . . . .	24
<b>5. Creando nuestra GAN</b>	<b>27</b>
5.1. Introducción . . . . .	27
5.2. Crear modelo base . . . . .	28
5.2.1. Descargar datos . . . . .	28
5.2.2. Ordenar datos . . . . .	28
5.2.3. Crear modelo base . . . . .	28
5.3. Entrenar GAN . . . . .	29

5.4. Generar imágenes Actinic Keratosis (AK) . . . . .	30
5.5. Incluir las imágenes generadas en el conjunto de datos original . . . . .	32
5.6. Modelo base con las imágenes generadas . . . . .	32
<b>6. Conclusiones y trabajo futuro</b>	<b>35</b>
6.1. Conclusiones . . . . .	35
6.2. Trabajo futuro . . . . .	37
<b>Bibliografía</b>	<b>39</b>
<b>Apéndices</b>	<b>45</b>
<b>A. Códigos</b>	<b>45</b>
A.1. Ordenar Datos . . . . .	45
A.2. Modelo Base . . . . .	49
A.3. Red GAN . . . . .	52
A.4. Generar imágenes . . . . .	58

# Índice de figuras

2.1. Ejemplo de convolución . . . . .	4
2.2. Ejemplo de detección de diagonales . . . . .	4
2.3. Funciones sigmoide y relu respectivamente . . . . .	5
2.4. Ejemplo de max-pooling . . . . .	5
2.5. Ejemplo de una arquitectura CNN . . . . .	6
2.6. Arquitectura del modelo clasificador VGG16 . . . . .	6
2.7. Arquitectura GAN a altos rasgos . . . . .	7
2.8. Imágenes de dormitorios generadas con una arquitectura GAN. . . . .	8
2.9. Arquitectura utilizada por NVIDIA para generar imágenes de caras de alta calidad . . . . .	8
2.10. Imagen generada a partir de un esbozo . . . . .	9
2.11. Imágenes generadas a partir de un texto . . . . .	9
3.1. Arquitectura para el procesamiento de datos propuesto por el artículo [1] .	14
3.2. Distribución del conjunto de datos con el que se entrenan los modelos de clasificación . . . . .	15
3.3. Arquitectura de la red generativa y la red discriminatoria propuestas .	15
3.4. Tabla descriptiva de la arquitectura de la red generativa y la red discriminatoria . . . . .	16
3.5. Arquitectura PGAN propuesta . . . . .	17
3.6. Arquitectura PGAN propuesta . . . . .	18
3.7. Arquitectura GAN propuesta . . . . .	18
4.1. Ejemplo del funcionamiento de una arquitectura GAN . . . . .	22
4.2. Arquitectura Gan . . . . .	23
4.3. Arquitectura de la red generativa. (Fuente: [2]) . . . . .	23
4.4. Arquitectura de la red discriminativa. (Fuente: [2]) . . . . .	24
5.1. Diagrama de bloque . . . . .	27
5.2. Acierto sobre épocas del modelo base . . . . .	29
5.3. Pérdida sobre épocas del modelo base . . . . .	29
5.4. Acierto sobre épocas del modelo base (tamaño imagen 64x64) . . . . .	30
5.5. Pérdida sobre épocas del modelo base (tamaño imagen 64x64) . . . . .	31
5.6. Imágenes del tipo cáncer AK . . . . .	31
5.7. Imágenes generadas por la red GAN . . . . .	31
5.8. Acierto sobre épocas del modelo base con imágenes AK generadas . . .	32
5.9. Pérdida sobre épocas del modelo base con imágenes AK generadas . . .	33

6.1.	Modelo base con los datos proporcionados por ISIC-2019 (izquierda) y	
	Modelo base con imágenes AK generadas . . . . .	36

# Índice de tablas

3.1. Análisis comparativo de la competición ISIC durante 2017-2019 . . . .	12
3.2. Tabla comparativa que refleja el balanceado del conjunto de datos proporcionados en la competición ISIC 2019 para realizar la tarea de clasificación. . . . .	20





# Capítulo 1

## Introducción

### 1.1. Motivación

La Inteligencia Artificial es uno de los términos de moda en la actualidad: máquinas capaces de aprender, a través de unos datos, patrones de comportamiento para luego crear un modelo, y así ser capaces de inferir conocimiento a través de datos nunca antes modelados.

Sin embargo, esto no es algo nuevo. Ya en la década de los sesenta se empezaba a escuchar sobre la Inteligencia Artificial. Aunque es en estos últimos años que, gracias a la potencia de procesamiento, nuevos algoritmos y, sobretodo, a la ingente cantidad de datos que poseemos, ha provocado un repunte significativo del uso de la inteligencia artificial en gran parte de las distintas áreas de la sociedad: comercio de acciones, robótica, cámaras de vigilancia, videojuegos, descubrimientos científicos, medicina, contrataciones, etc.

Viendo que la Inteligencia Artificial abarca tantos campos, y que día a día crece su alcance, la motivación de este Trabajo Fin de Grado es adentrarse en un campo específico de la misma: la detección de cáncer de piel.

El cáncer de piel es uno de los tumores con mayor frecuencia en la sociedad. Es por ello que se hace vital, para mejorar la calidad de vida humana, tener herramientas capaces de detectar de manera acertada si una persona padece cáncer de piel y de qué tipo. Sin embargo, uno de los grandes problemas en este campo es la falta de datos, en este caso imágenes. Por esta razón, con este ensayo se busca aprender cómo una red neuronal puede ser capaz de crear imágenes, nunca antes vistas, que simulen un tipo de cáncer de piel.

### 1.2. Objetivos

El objetivo de este Trabajo Fin de Grado es aprender, primeramente, en qué consiste una red neuronal generativa adversaria (a partir de ahora se utilizará el acrónimo GAN para referirnos a ella), para luego, entender cómo se puede aplicar esta arquitectura a un problema de la vida real, como lo es la detección de cáncer de piel.

Una vez se haya comprendido el funcionamiento de una GAN, cuáles son los elementos que la componen, los hiper parámetros que se pueden modificar para obtener un mejor resultado, entre otros, este trabajo se enfocará en una competición anual, Skin Lesion Analysis Towards Melanoma Detection. Se trata de una competición a nivel mundial donde, dado una serie de imágenes, el concursante ha de ser capaz de crear un modelo que prediga si estas imágenes contienen cáncer de piel y de qué tipo.

Utilizando el conjunto de datos proporcionados por la competición, por medio de transfer learning, se creará un modelo capaz de clasificar entre los distintos tipos de cáncer. Una vez este modelo esté entrenado se intentará, por medio de una arquitectura GAN, crear una red que genere nuevas imágenes. El objetivo de esto es ampliar el conjunto de datos proporcionados y ver así si el modelo de clasificación creado anteriormente mejora.

### 1.3. Organización de la memoria

Esta memoria consta de los siguientes capítulos:

- **Capítulo 1** Introducción.
- **Capítulo 2** Estado del arte.
- **Capítulo 3** ISIC.
- **Capítulo 4** GAN.
- **Capítulo 5** Creando nuestra GAN.
- **Capítulo 6** Conclusiones y Trabajo Futuro.

# Capítulo 2

## Estado del arte

### 2.1. Introducción

Una de las áreas de la Inteligencia Artificial es el aprendizaje profundo, o también conocido como Deep Learning. Dentro del Deep Learning existe una infinidad de arquitecturas que buscan dar solución a distintas problemáticas.

Este Trabajo Fin de Grado se centrará en dos arquitecturas específicamente: redes neuronales convolucionales (CNN, convolutional neural network) y redes neuronales generativas adversarias (GAN, generative adversarial network). A continuación, se dará una pequeña introducción sobre las mismas. En el siguiente capítulo se hablará de manera más específica de la arquitectura GAN, ya que aquí solo se dará una pincelada de la misma.

### 2.2. Redes Neuronales Convolucionales (CNN)

Una red neuronal convolucional es un algoritmo de Deep Learning utilizado para el análisis de imágenes y vídeos. Con esta arquitectura, el modelo aprende distintos tipos de patrones y características comunes dentro un conjunto de datos. Una vez el modelo ha aprendido estos patrones es capaz de realizar distintos tipos de clasificaciones, detección de objetos o segmentación. Un ejemplo es el modelo ResNet[3], ganador de una competición de clasificación de distintas imágenes en el año 2015, obteniendo un error del 3,57 % (se buscaba clasificar de manera acertada entre 1000 categorías).

La red convolucional es una red multi-capa, donde cada capa contiene 3 elementos fundamentales:

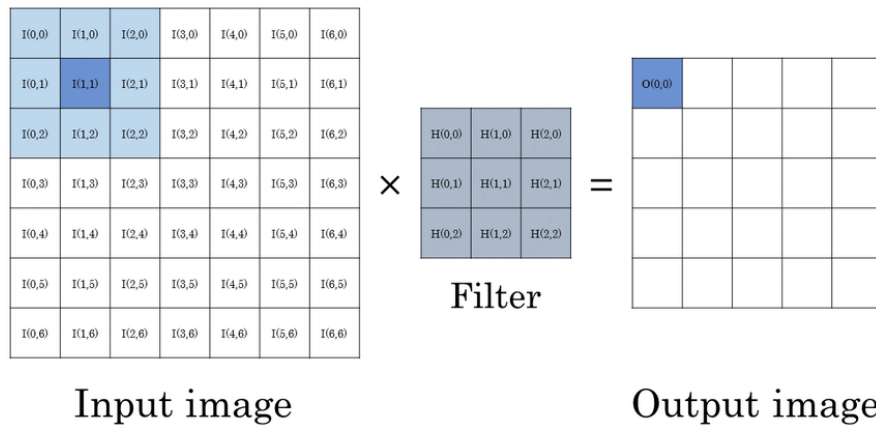
- Convolución
- Filtrado no lineal
- Redimensionado espacial

### 2.2.1. Convolución

Consiste en la aplicación de un filtro (normalmente de tamaño 3x3 o 5x5) a la imagen para obtener características locales. Una vez aplicada la convolución, el tamaño y la profundidad de la imagen resultante será diferente.

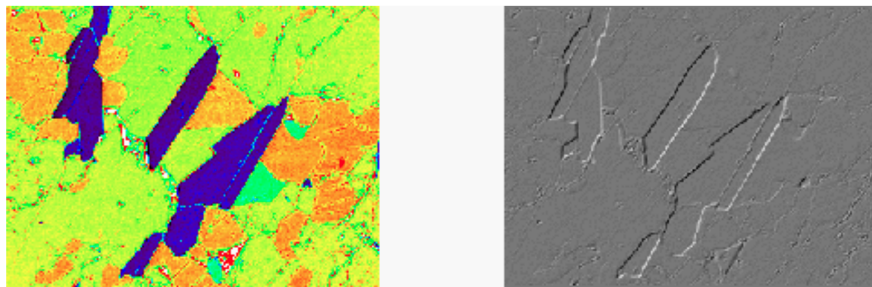
Un filtro es una matriz con una serie de valores con los que se realiza un mapeo de la imagen, donde se multiplica cada elemento del filtro con cada elemento de la imagen y luego se realiza una suma de todos los elementos resultantes para obtener un único valor.

A continuación se puede observar un ejemplo de convolución:



Un filtro de tamaño 3x3 se aplica a una imagen y se obtiene una imagen resultante. En este caso, la imagen resultante es más pequeña. (Fuente: [4])

Figura 2.1: Ejemplo de convolución

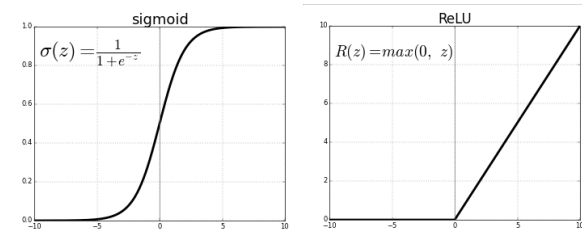


Aplicando el filtro  $\begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}$  se puede obtener las diagonales que aparecen en la imagen. (Fuente: [5])

Figura 2.2: Ejemplo de detección de diagonales

### 2.2.2. Filtrado no lineal

Una vez finalizada la convolución se realiza una transformación de los datos obtenidos, por medio de una función no lineal. Se puede utilizar una función sigmoideal,  $\frac{1}{1+e^{-z}}$ , o una función relu,  $Relu(x) = max(0, x)$  donde se devuelve x si x es mayor que 0, sino 0.

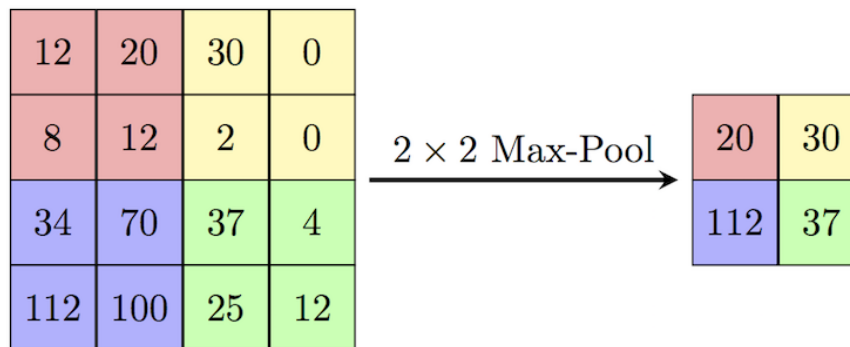


A mano izquierda se puede observar el comportamiento de la función sigmoide, mientras que a mano derecha la función relu.

Figura 2.3: Funciones sigmoide y relu respectivamente

### 2.2.3. Redimensionado espacial

Una vez se han realizado los pasos anteriores, al algoritmo le interesa únicamente los datos más significativos. Esto se realiza mediante un pooling (max-pooling, average-pooling, etc.). Con esto se consigue una reducción de parámetros en la red y la invarianza espacial.

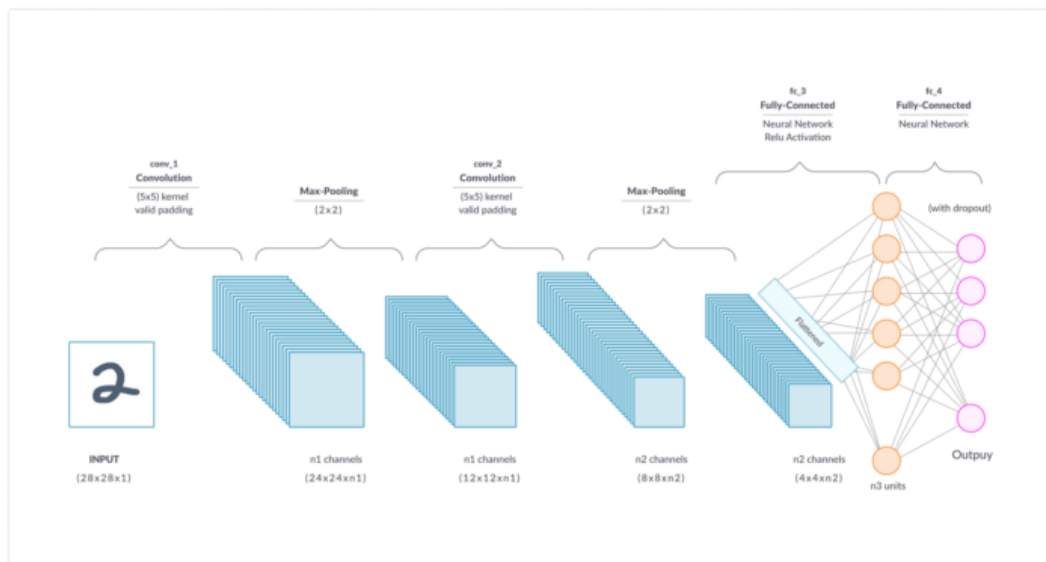


Este es un ejemplo de cómo funciona un max-pooling. Dado un filtro de tamaño 2x2 se obtiene los valores máximos de cada sección. De esta manera, sólo quedan los valores más significativos, y también así se reduce el tamaño de parámetros con los que la red trabaja.

Figura 2.4: Ejemplo de max-pooling

Una vez finalizadas estas tres operaciones (convolución, filtrado y redimensionado), aparecen las capas completamente conectadas (layers fully connected) donde cada neurona está conectada con todas las neuronas de la capa posterior, de ahí el nombre fully connected.

Una vez obtenidos los patrones y las características extraídas, es el momento de realizar la clasificación. De ahí la importancia de las capas fully connected. Cabe mencionar que entre cada capa completamente conectada hay una función softmax, que da una probabilidad sobre lo que la neurona debe predecir.

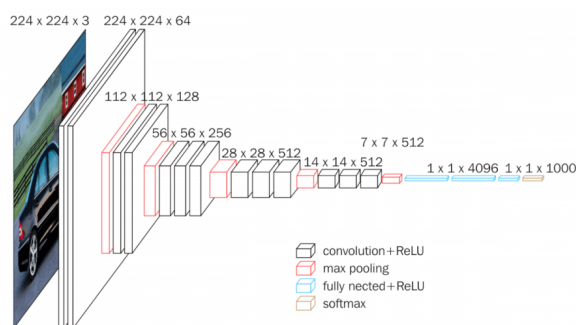


Aquí podemos ver desde la convolución, pasando por el max-pooling a cómo las últimas capas están completamente conectadas. Es esto consiste una arquitectura CNN. (Fuente: [6])

Figura 2.5: Ejemplo de una arquitectura CNN

A lo largo de estos últimos años, este tipo de arquitectura de red neuronal ha tenido un gran avance, especialmente en el campo de la clasificación de imágenes. En la documentación de Keras<sup>1</sup>, una librería escrita en Python para trabajar con redes neuronales, podemos encontrar distintos modelos de clasificación con muy buenos resultados. Entre ellos podemos encontrarnos con Xception, VGG16, Inception, NasNet, entre una amplia variedad.

A continuación se puede ver la arquitectura VG16, donde se visualizan los elementos de convolución, pooling, capas completamente conectadas y la función softmax. Tras estos elementos, se obtiene una red que clasifica entre 1000 categorías distintas.



Arquitectura del modelo VGG16, consistente en varias convoluciones, funciones no lineales tipo relu, max-pooling y terminando con una función softmax para clasificar entre 1000 categorías. (Fuente: [7])

Figura 2.6: Arquitectura del modelo clasificador VGG16

<sup>1</sup><https://keras.io/applications/>

## 2.3. Redes Generativas Adversarias (GAN)

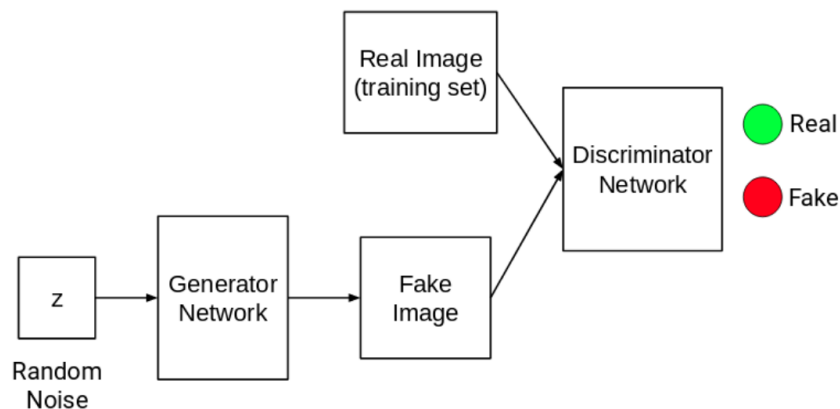
Las red generativa adversaria es una arquitectura de Deep Learning que busca generar un nuevo dato en base a lo que ha aprendido. Utiliza dos redes neuronales que compiten una con la otra, es decir, son adversarias (de ahí su nombre, GAN). Estas redes luchan entre sí para ver quién gana y quién pierde, esto es conocido como el juego de suma cero en teoría de juegos. Ambas redes juegan un papel importante: por un lado tenemos una red que genera y por otro una red que discrimina.[8]

- Red Generativa: se encarga de crear datos que parezcan reales, por ejemplo crear una cara.
- Red Discriminativa: decide si el dato generado es real o no.

Para entrenar esta arquitectura se necesita una buena cantidad de datos, ya que el discriminador debe aprender y ser capaz de distinguir entre lo real y lo artificial (creado por la red generativa).

Primero, se entrena la red discriminativa para que sea capaz de diferenciar entre los datos verdaderos y falsos. Una vez se tiene esto como punto de partida, se ejecutan ambas redes. Al principio será fácil para la red discriminativa acertar en su predicción, pero con el paso de las épocas los datos creados por la red generativa serán mejores, produciendo así que la tarea de la red discriminativa sea más complicada. En medio de esta competición, ambas redes aprenderán y mejorarán simultáneamente.

A continuación se puede ver un ejemplo de una arquitectura GAN a altos rasgos.



Se puede observar cómo ambas redes trabajan conjuntamente para crear la arquitectura GAN. (Fuente: [9])

Figura 2.7: Arquitectura GAN a altos rasgos

Uno de los primeros usos de la arquitectura GAN fue el generar dormitorios a partir de unas imágenes reales dadas de distintos dormitorios. Podemos verlo en la figura de a continuación.

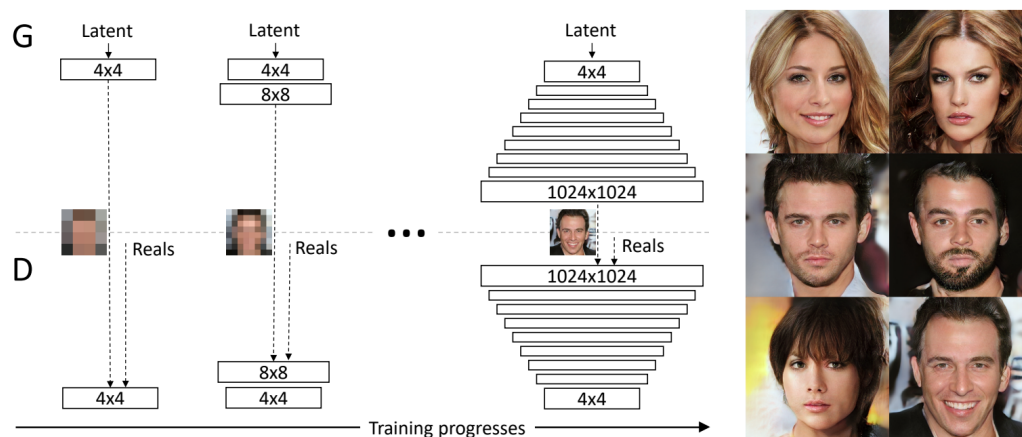




Se puede distinguir distintas imágenes de dormitorios generados por una arquitectura GAN. Es el comienzo de una nueva arquitectura dentro de Deep Learning. (Fuente: [10])

Figura 2.8: Imágenes de dormitorios generadas con una arquitectura GAN.

Sin embargo, como se puede apreciar las imágenes tienen una baja calidad. Posteriormente, NVIDIA publicó un artículo donde mitiga este problema creando caras muy realistas (lo consigue partiendo de un generador/discriminador para imágenes muy pequeñas y, una vez éste es muy bueno, va aumentándole el tamaño progresivamente).



La propuesta de NVIDIA es tener un generador/discriminador suficientemente bueno para una imagen 4x4 y de ahí pasar a 8x8 y sucesivamente hasta conseguir una imagen de tamaño 1024x1024. A la derecha se ven ejemplos de caras creadas de manera artificial. (Fuente: [11])

Figura 2.9: Arquitectura utilizada por NVIDIA para generar imágenes de caras de alta calidad

Otra aplicación bastante interesante de esta arquitectura es la creación de una imagen a partir de un boceto. En el artículo Image-to-Image Translation with Conditional



Adversarial Network de la Universidad de Berkeley[12] se ven distintas aplicaciones de este concepto. Crear un boceto y a partir del mismo generar la imagen realista del mismo, oscurecer una imagen o viceversa son algunas de ellas.



A través de un simple esbozo, el modelo es capaz de generar una imagen muy parecida al esbozo realizado. (Fuente: [12])

Figura 2.10: Imagen generada a partir de un esbozo

Una última aplicación, de una gran variedad de aplicaciones que se pueden encontrar, es traducir texto escrito en frases sencillas a imágenes. Por ejemplo, se puede escribir “this small bird has a short pointy orange beak and white belly” (este pequeño pájaro tiene un pequeño y puntiagudo pico naranja y una barriga blanca) y obtener la siguiente imagen:



Ejemplos de imágenes generadas a partir de una frase descriptiva. La primera imagen contando por arriba a la izquierda es una imagen real, mientras que el resto son imágenes generadas. (Fuente: [13])

Figura 2.11: Imágenes generadas a partir de un texto

Vistas estas aplicaciones cabría preguntarse: ¿puede esta arquitectura ser de ayuda para mejorar la capacidad de clasificación en los problemas con escasez de datos? En

particular, ¿puede la arquitectura GAN ayudar en cierta manera para mejorar un clasificador de distintos tipos de cáncer de piel?

Para responder a esta pregunta, primero se introducirá la ISIC Challenge y se verá qué se ha estado haciendo hasta el momento.

# Capítulo 3

## Competición ISIC

### 3.1. Introducción

Cuando se habla de ISIC Challenge<sup>1</sup> se está refiriendo a una competición a nivel internacional que nace en el año 2016. El objetivo de la competición es crear modelos capaces de clasificar entre distintos tipo de cáncer de piel, además de tratar otros aspectos como puede ser la segmentación del cáncer en cada imagen. Desde que comenzó hasta ahora, el conjunto de datos proporcionado por la organización se ha ido ampliando y enriqueciendo.

Cabe mencionar que en los años 2016, 2017 y 2018 la competición consistía en segmentar la lesión, detectar y localizar patrones y clasificar el cáncer. En 2019 se redujo a dos tareas: 1) clasificar imágenes demoscópicas sin metadatos y 2) clasificar imágenes adicionales con metadatos.

En este Trabajo Fin de Grado se utiliza el conjunto de datos proporcionado en el año 2019. En el concurso de ese año se buscó clasificar entre ocho tipos de cánceres: Melanoma, Melanocytic nevus, Basal cell carcinoma, Actinic keratosis, Bening keratosis, Dermatofibroma, Vascular lesion, Squamous cell carcinoma o ninguno de los cánceres anteriores.

Para esta labor la organización proporcionó varios conjuntos de datos. El principal[14] consiste en 25,331 imágenes demoscópicas; luego, está el dataset HAM10000[15] que contiene imágenes de tamaño 600x450 centradas alrededor de la lesión y, el último dataset, BCN 20000[16], contiene imágenes de tamaño 1024x1024 de difícil utilización ya que las lesiones están en localizaciones poco comunes.

### 3.2. Tabla comparativa

Una de las tareas previas para ver cómo se puede aplicar una arquitectura GAN en este problema de clasificación es la de investigar qué es lo que se ha estado haciendo en estos años previos en la competición. Es por ello que a continuación se puede observar una tabla comparativa con su análisis respectivo. La comparativa se realiza para los años 2017, 2018 y 2019 y, posteriormente se detalla la composición de la tabla.

---

<sup>1</sup><https://www.isic-archive.com/#!/topWithHeader/wideContentTop/main>

- Equipo: nombre del equipo participante.
- Año: Año en el que se lleva a cabo la ejecución.
- Posición (pos): posición en la que quedó el equipo en la tarea de clasificación.
- Arquitectura: se apunta de qué tipo de arquitectura se trata.
- Transfer Learning (TL): se especifica si el modelo está utilizando conocimiento adquirido por modelos entrenados para otra tarea.
- Reentrenamiento de cero (0): se ve si se entrenan los pesos del modelo desde cero (lo contrario a transfer learning).
- Ensemble model (Ens): utiliza múltiples modelos para luego dar una probabilidad de la mejor predicción.
- Dataset externo (Ext): entrena el modelo únicamente con los datos proporcionados o utiliza datos externos para esta tarea.

A continuación se puede observar los datos obtenidos de la comparativa (estos datos se han obtenido viendo y leyendo el histórico de la competición).

Equipo	Año	Pos	Arquitectura	TL	0	Ens	Ext
DAISYLab[17]	2019	1º	Ensemble EfficientNet	x		x	x
DYsionAI[18]	2019	2º	Deep CNN Ensemble	x		x	
AlmageLab[19]	2019	3º	ResNet-152, DenseNet-201, SeResNext-50, otros	x		x	
DermanCode[20]	2019	4º	SENet, ResNet-50/VGG-16/otros	x		x	
Nurithm Labs[21]	2019	5º	Densenet-161		x		x
MetaOptima[22]	2018	1º	DPN-92, Resnet-152, Densenet-161, otros	x		x	x
MetaOptima[22]	2018	2º	Meta Ensemble	x		x	x
MetaOptima[22]	2018	3º	DPN-92(5k)	x		x	x
DAISYLabs[23]	2018	4º	Densenet, SENet y ResNetXt	x		x	
Sun Yat-sen[24]	2018	5º	SENet y PNASNet-5-Large	x		x	
Matsunaga[25]	2017	1º	ResNet	x		x	x
monty python[26]	2017	2º	CNN con conocimiento experto	x			
RECOD Titans[27]	2017	3º	VGG-16	x		x	
popleyi[28]	2017	4º	ResNets, VGGNet	x		x	
Xulei Yang[29]	2017	5º	GoogleNet y U-Net		x		

Tabla 3.1: Análisis comparativo de la competición ISIC durante 2017-2019

Las observaciones obtenidas tras realizar la tabla comparativa de las competiciones del 2017 al 2019 son las siguientes:

1. Prácticamente todos los participantes usaron la técnica de combinar distintas arquitecturas, es decir, utilizaron ensemble model.
2. La mayoría no partió desde cero a la hora de entrenar sus modelos, sino que utilizaron transfer learning. Los participantes que utilizaron esta técnica emplearon ImageNet.

3. La mitad de los participantes utilizó un dataset propio o externo. Una conclusión que se podría obtener a simple vista es que no es determinante utilizar un dataset externo para lograr posicionarte en el top 5 de la competición.
4. Si se observan las arquitecturas utilizadas, se repiten varias como ResNet, SEnet, VGG o DenseNet.
5. Ningún participante del top 5 ha utilizado la arquitectura GAN(generative adversarial network).

Hechas las pertinentes observaciones, ¿por qué no se utilizan las redes generativas adversarias? ¿Ha habido algún equipo que haya utilizado esta arquitectura para abordar este problema? ¿Todavía no tiene una capacidad determinista para afrontar este tema de problemas de clasificación?

A continuación se analizan distintos artículos de equipos que sí utilizaron las GAN para este tipo de problemas pero que no estuvieron entre los mejores colocados en el ranking final.

### 3.3. Análisis de artículos que SÍ utilizaron GAN

Las redes generativas adversarias suelen, una vez leídos los artículos de los equipos que sí emplean las GAN, ser utilizadas como una herramienta para data augmentation. Dicho en otras palabras, se utiliza esta arquitectura como una técnica para aumentar el conjunto de datos del que se dispone, para así entrenar el modelo de clasificación y obtener un mejor acierto a la hora de predecir qué tipo de cáncer se trata en la imagen dada.

En el artículo “Towards Automated Melanoma Detection with Deep Learning: Data Purification and Augmentation”[1] se utilizan dos DCGANs (Deep Convolutional Generative Adversarial Networks) para generar 350 imágenes de melanoma y 750 de seborrheic keratosis, dos categorías que estaban bastante desbalanceadas en el dataset proporcionado en la competición del ISIC de 2017.

Otro artículo, “Data Augmentation for Skin Lesion using Self-Attention based Progressive Generative Adversarial Network”,[30] utiliza también la arquitectura GAN para aumentar el conjunto de datos y así ser capaces de mejorar el rendimiento de la clasificación. En este escrito, además de comentar que se utilizó PGAN (Progressive Generative Adversarial Network), muestran una mejora en la precisión del 2,8 % pasando de un 67,3 % a un 70,1 %.

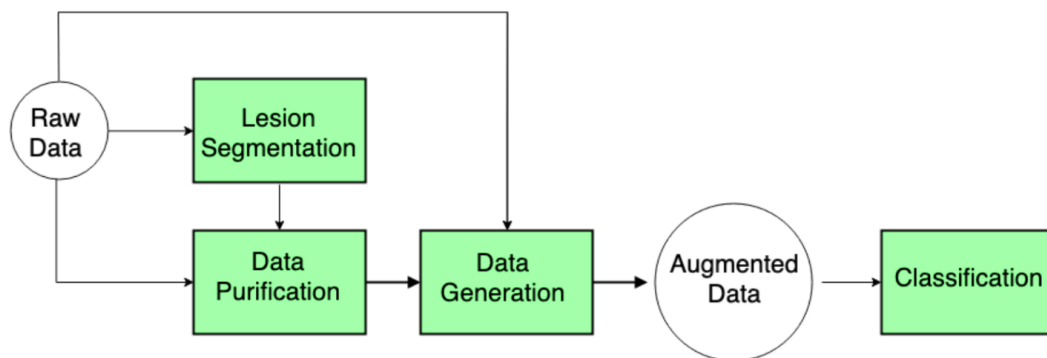
Una vez más, en el artículo “Skin Lesion Analysis Towards Melanoma Detection Using Generative Adversarial Network”[31] aportado por el equipo SIBET CAS en la competición del 2019, se menciona la dificultad a la hora de crear buenos modelos de clasificación con un dataset desbalanceado. Es por ello que para afrontar esta dificultad propone utilizar SSGAN (Semi Supervised Generative Adversarial Network) para así crear nuevas imágenes y balancear mejor el dataset dado.

Leídos los artículos anteriores se puede observar una clara tendencia de la utilización de la arquitectura GAN al campo de la aumentación del dataset propuesto. Si un conjunto de datos está desbalanceado podemos aplicar esta arquitectura para generar nuevos datos y así equilibrarlo. Una vez se realiza este paso uno es capaz de anteponerse a esta dificultad y crear un modelo que discrimine entre las diversas categorías de una mejor manera.

A continuación se analiza de manera profunda cada uno de estos tres artículos mencionados anteriormente.

### 3.3.1. Towards Automated Melanoma Detection with Deep Learning: Data Purification and Augmentation

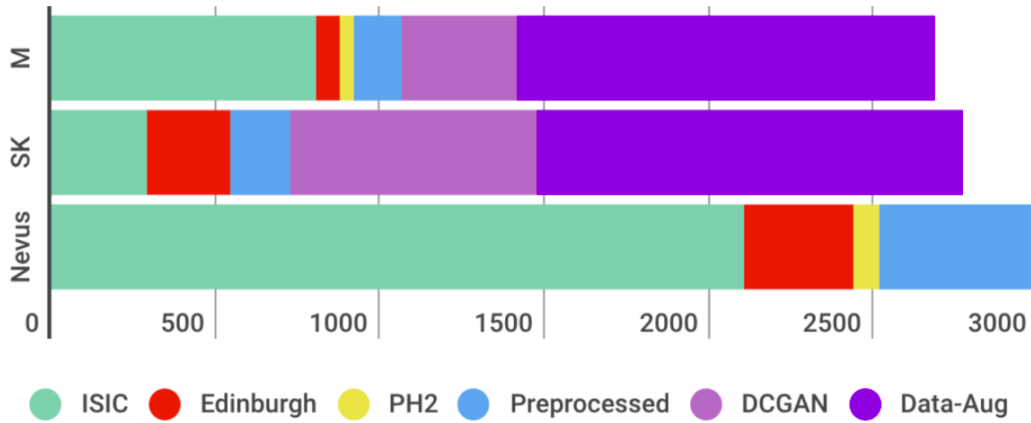
En este artículo[1] se utiliza la arquitectura GAN dentro del área de preparación de los datos. Los datos pasan por tres estados: segmentación, purificación y aumentación de los mismos. En la siguiente figura se observa el aspecto que tiene el área de preparación de datos.



Se observa que antes de que se cree un modelo clasificador, los datos primero pasarán por los estados de segmentación, purificación y aumentación para generar un mejor conjunto de datos con los que el modelo aprenderá a modelar. (Fuente: [1])

Figura 3.1: Arquitectura para el procesamiento de datos propuesto por el artículo [1]

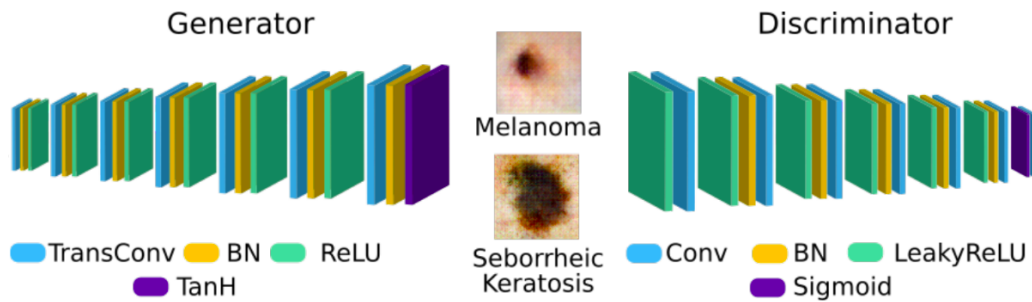
Uno de los grandes problemas que se encuentran los participantes es que el conjunto de datos proporcionado está desbalanceado, provocando que el modelo esté sesgado a la categoría de mayor presencia. La solución que se propone en este artículo es utilizar distintos métodos, entre ellos la generación sintética de nuevos datos, es decir, crear nuevas imágenes.



En esta imagen se observa de dónde se obtienen los datos para las categorías melanoma, nevus y seborrheic keratosis. Los datos provienen de distintos conjuntos de datos: ISIC, Edinburgh y PH2, del preprocesado de datos, aumentación y de la arquitectura DCGAN. Se visualiza que para melanoma y seborrheic keratosis un buen porcentaje de los datos son generados. (Fuente: [1])

Figura 3.2: Distribución del conjunto de datos con el que se entrenan los modelos de clasificación

Referente a la estructura de la arquitectura que utilizan es la siguiente:



La red generativa parte un tamaño de imagen pequeña para, progresivamente, ir aumentándola hasta alcanzar el tamaño de las imágenes proporcionadas en el conjunto de datos de la competición. En cada capa se realiza TransConv (Transposed Convolutions), que básicamente es lo opuesto a una convolución (en una convolución se reduce el tamaño de la entrada, mientras que aquí se aumenta), BN (batch normalization, técnica para normalizar la activación en las capas intermedias) y utiliza la función de activación no lineal Relu. En la última capa utiliza una tangente hiperbólica como función de activación.

Por el otro lado, la red discriminativa realiza el proceso opuesto, cada vez busca entradas más pequeñas, por medio de convoluciones, BN y utilizando las funciones de activación LeakyRelu y sigmoide. (Fuente: [1])

Figura 3.3: Arquitectura de la red generativa y la red discriminativa propuestas

Si se quiere ahondar un poco más sobre cada capa, los autores proporcionan una matriz donde explican la configuración de cada capa.

	Layer	Output Size	Kernel	Stride	Padding
Generator	TransConv	$256 \times 4 \times 4$	$4 \times 4$	1	0
	TransConv	$128 \times 8 \times 8$	$4 \times 4$	2	1
	TransConv	$64 \times 16 \times 16$	$4 \times 4$	2	1
	TransConv	$32 \times 32 \times 32$	$4 \times 4$	2	1
	TransConv	$16 \times 64 \times 64$	$4 \times 4$	2	1
	TransConv	$8 \times 128 \times 128$	$4 \times 4$	2	1
	TransConv	$3 \times 256 \times 256$	$4 \times 4$	2	1
Discriminator	Conv	$16 \times 128 \times 128$	$4 \times 4$	2	1
	Conv	$32 \times 64 \times 64$	$4 \times 4$	2	1
	Conv	$64 \times 32 \times 32$	$4 \times 4$	2	1
	Conv	$128 \times 16 \times 16$	$4 \times 4$	2	1
	Conv	$256 \times 8 \times 8$	$4 \times 4$	2	1
	Conv	$512 \times 4 \times 4$	$4 \times 4$	1	1
	Conv	$1 \times 1 \times 1$	$4 \times 4$	1	0

Se observa la descripción de la configuración de cada capa, con su salida, kernel que se utiliza, stride y padding tanto para la red generadora como discriminadora. (Fuente: [1])

Figura 3.4: Tabla descriptiva de la arquitectura de la red generativa y la red discriminadora

Algunos datos que se deben tener en cuenta, relacionado con la manera en la que se configuraron las redes neuronales:

- La función de pérdida utilizada fue: binary cross entropy loss.
- El optimizador empleado fue: Adam optimizer con una tasa de aprendizaje de  $2e - 4$  y valores beta de 0.5 y 0.999 .
- Utilizan la técnica de estabilización (comenzar por una imagen pequeña y progresivamente ir aumentando su tamaño para obtener imágenes generadas de calidad).

También destacar que utilizan un modelo pre-entrenado, RESNET-50, con imágenes del conjunto de datos ImageNet. Los resultados que obtienen son los siguientes para la clasificación del tipo de cáncer Melanoma:

- NO utilizando preprocesado de datos (segmentación, generación y aumentación): 80,5 % de acierto.
- Sí utilizando preprocesado de datos: 88 % de acierto.

En conclusión, aplicando la arquitectura GAN, se produce una mejora de casi el 8 % en la capacidad clasificatoria del modelo.



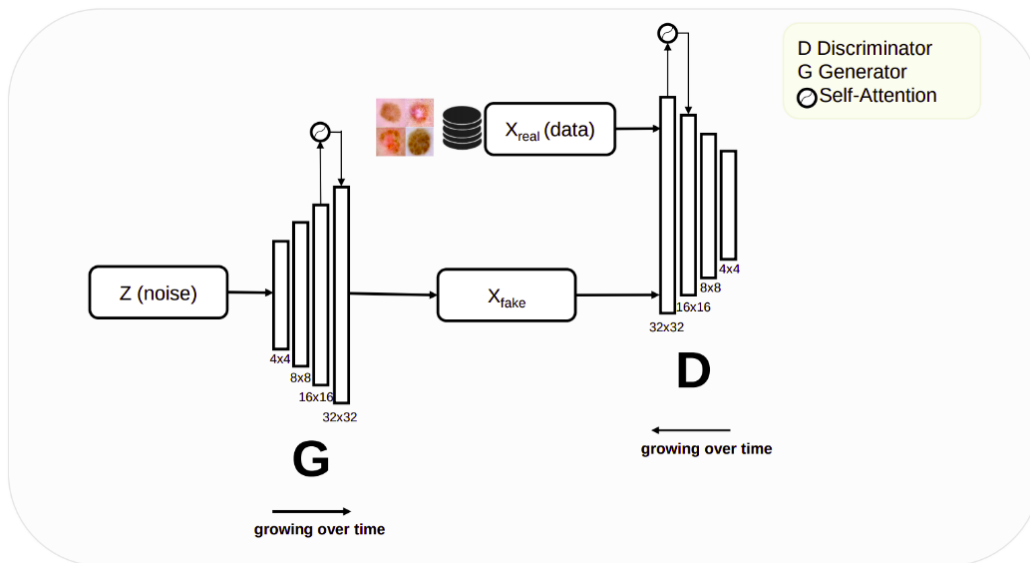
### 3.3.2. Data Augmentation for Skin Lesion using Self-Attention based Progressive Generative Adversarial Network

Este artículo[30] propone una solución a los problemas derivados de la utilización, en exclusiva, de la aumentación de datos. Principalmente, la aumentación tiene dos problemas: 1. Se necesita un conocimiento experto del conjunto de datos con el que se trabaja y 2. No funciona para cualquier conjunto de datos (por ejemplo, si queremos clasificar números escritos a mano, no tiene mucho sentido aplicar rotación en el número 6 ya que el modelo lo confundiría con el número 9).

La solución a los problemas derivados de la aumentación de datos consiste en generar nuevas imágenes por medio de varias arquitecturas GAN (cada una de ellas contiene ligeras variaciones<sup>2</sup>). En concreto utiliza tres:

1. PGAN - progressive generative adversarial network
2. APGAN - attention progressive growing of GAN
3. APGAN + TTUR - attention progressive growing of GAN with two time update rule (imbalanced learning rate)

A continuación se puede ver una panorámica de la red APGAN+TTUR.

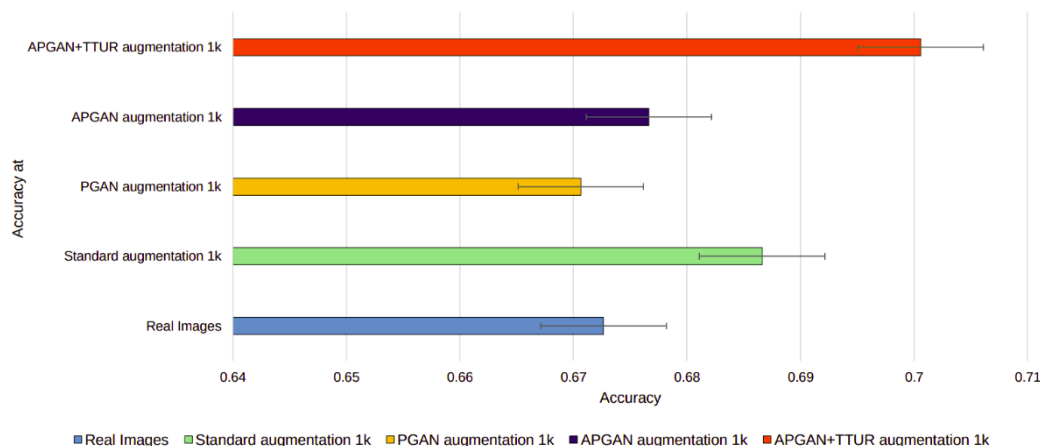


Se observa a G (red generativa) y D (red discriminativa) con sus diferentes capas y tamaños. (Fuente: [30])

Figura 3.5: Arquitectura PGAN propuesta

Para la clasificación utiliza el modelo ResNet-18 preentrenado con imágenes de ImageNet. Los resultados obtenidos utilizando estos diferentes modelos clasificatorios son los siguientes:

<sup>2</sup>En este proyecto no se tratan los distintos modelos de una arquitectura GAN, ya que se enfoca en la arquitectura clásica.



Se observa el distinto porcentaje de acierto utilizando los distintos modelos, aumentación estándar o únicamente las imágenes reales. (Fuente: [30])

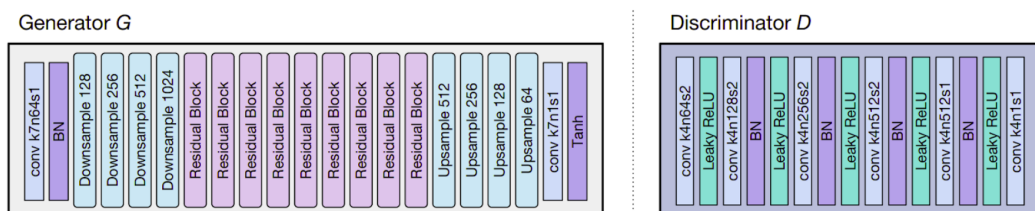
Figura 3.6: Arquitectura PGAN propuesta

Utilizando la arquitectura GAN para generar nuevas imágenes, y así ser capaces de crear un modelo que pueda predecir mejor, obtiene un buen resultado. Se produce una mejora del 2,8 % de acierto, pasando del 67,3 % al 70,1 %.

### 3.3.3. Skin Lesion Analysis Towards Melanoma Detection Using Generative Adversarial Network

En este artículo[31] se utiliza la arquitectura GAN para paliar una necesidad: falta de datos. Aunque hay datos, imágenes en esta ocasión, no son suficientes para crear modelos altamente precisos. Es por ello que se necesita aumentar estos datos, pero este proceso para incrementar el conjunto de datos es caro y extenso en tiempo. La solución es, por tanto, generar imágenes nuevas.

Para ello utilizan la arquitectura PGAN y como base pix2pixHD.[32] La configuración de la red generativa y la red discriminativa es la siguiente:



Se observa la configuración de la arquitectura propuesta: podemos ver cómo utiliza batch normalization, convoluciones, de-convoluciones y distintas funciones de activación. (Fuente: [31])

Figura 3.7: Arquitectura GAN propuesta

Además de utilizar la arquitectura GAN emplea un mapeo semántico (una imagen donde cada pixel tiene el valor de la clase objeto) y una instancia semántica (una

### 3.4. *¿POR QUÉ UTILIZAR GAN EN LA DETECCIÓN DE CÁNCER DE PIEL?*<sup>19</sup>

imagen donde los píxeles combinan información entre su instancia y la clase objeto) para obtener mejores resultados.

Sin embargo, en este artículo es donde menos mejora se produce: sólo hay un 1 % de diferencia (se pasa de un 83 % de AUC a un 84 %) respecto a no utilizar esta técnica para aumentar el conjunto de datos.

A continuación se trata el por qué de utilizar la arquitectura GAN en la detección de cáncer de piel, una vez analizados los artículos anteriores.

### 3.4. **¿Por qué utilizar GAN en la detección de cáncer de piel?**

Una de las grandes ventajas que se tiene a día de hoy, respecto al pasado, son los datos: se posee una cantidad ingente de información. Pero con esta situación actual, subyacen dos problemas que se repiten una y otra vez: 1) no todo lo que se recopila es útil, usable, no sesgado, etc. y 2) no hay datos de áreas específicas, ya que hasta ahora no se veía la necesidad de recopilar. Un ejemplo de ello son las imágenes en alta resolución de distintos tipos de cáncer de piel.

Es aquí donde se hace necesario un preprocesado de los datos. Como consecuencia, se obtiene que la muestra de datos resultante es pequeña o está descompensada. Éste es el caso con el conjunto de datos proporcionados por la competición ISIC del año 2019.

Por un lado, hay 25.331 imágenes de ocho distintas categorías. Puede parecer una cantidad importante de datos, pero al trabajar con redes neuronales profundas mientras más datos (entendiendo que son datos de calidad) mejor modelo predictivo se obtendrá.

Por otro lado, si se analiza el conjunto de datos proporcionado se puede ver que está completamente desbalanceado, creando por consiguiente un gran sesgo y una mala generalización del modelo clasificatorio.

A continuación, se puede encontrar una tabla con el porcentaje de cada categoría sobre el total de datos.

Tipo de cáncer de piel	Porcentaje
Melanoma	17,85 %
Melanocytic nevus	50,83 %
Basal cell carcinoma	13,12 %
Actinic keratosis	3,42 %
Benign keratosis	10,37 %
Dermatofibroma	0,94 %
Vascular lesion	0,99 %
Squamous cell carcinoma	2,48 %
Total	100 %

Tabla 3.2: Tabla comparativa que refleja el balanceado del conjunto de datos proporcionados en la competición ISIC 2019 para realizar la tarea de clasificación.

Se observa que hay una categoría predominante, Melanocytic nevus, luego le siguen tres categorías con más de un 10 % de presencia en la muestra. Pero también se observa cómo hay cuatro categorías que no llegan al 4 % de presencia. Estamos ante un claro ejemplo de una muestra desbalanceada.

Obtener nuevas imágenes, de aquellas categorías que están poco representadas en el conjunto de datos original, implicaría que nuestro modelo no aprendiese sesgado por una o varias categorías predominantes. Sino por el contrario, se obtendría un modelo capaz de generalizar adecuadamente. Y a día de hoy se puede conseguir esto, es decir, con la arquitectura GAN se es capaz de generar imágenes sintéticas nunca antes vistas.

Aunque salta a la vista un claro interrogante, si es tan bueno y se es capaz de paliar estos problemas derivados de la falta de datos, ¿por qué no lo utilizaron todos los equipos? ¿Por qué ninguno de los clasificados en el top 5 de los distintos años en los que se realizó la competición lo utilizaron? Aunque la arquitectura GAN lleva un par de años y sigue desarrollándose día a día, todavía estamos en una fase de investigación y mejora de la misma. Es por ello que hay una gran necesidad de realizar pruebas y experimentos, y ver hasta dónde se es capaz de llegar y añadir valor a lo que ya se está realizando en distintas partes del mundo.

No obstante, cabe recordar que el objetivo de este trabajo es adentrarse en el funcionamiento de una red generativa adversaria. En ningún momento se intenta proponer o investigar alguna ramificación desconocida, ya que se está ante un proyecto introductorio.

# Capítulo 4

## Red Generativa Adversaria

### 4.1. Introducción

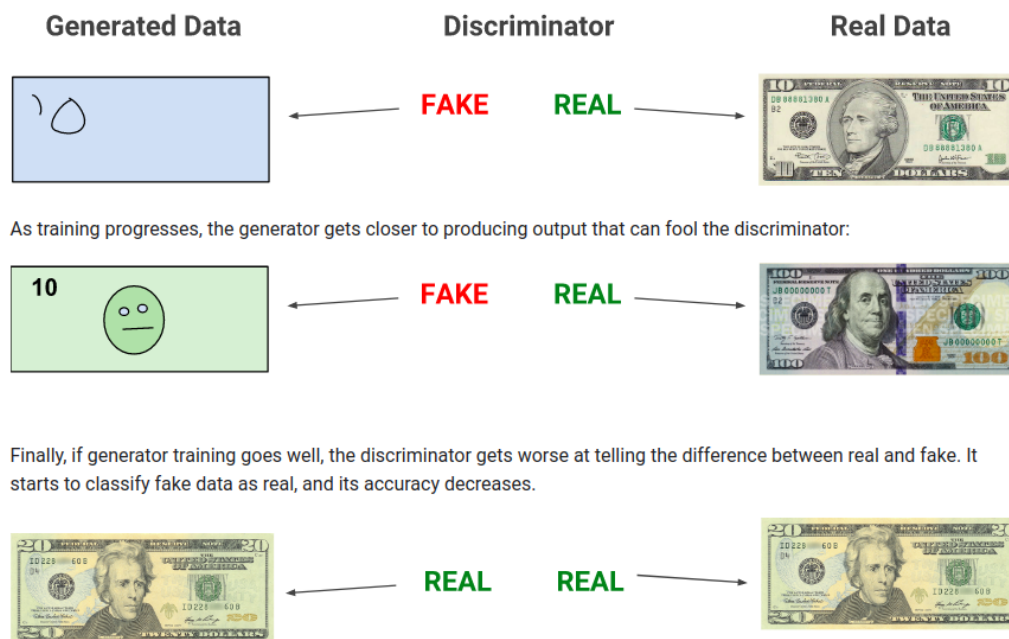
En este capítulo se explica qué es una red generativa adversaria. Aunque ya se ha hablado al respecto en el capítulo del Estado del Arte, en esta sección se estudiará en mayor profundidad. Se verán cuáles son los elementos que componen esta red y los distintos tipos de hiper parámetros que permiten configurar de distintas maneras la red.

### 4.2. Definición

Generative Adversarial Network (GAN), o en español red generativa adversaria, es una arquitectura de red neuronal dedicada a los problemas de aprendizaje no supervisado. Este tipo de red neuronal fue introducido por Ian Goodfellow en 2014. Este tipo de arquitectura está formado por dos redes neuronales profundas, las cuales cumplirán una función específica. Éstas redes se denominan: red generativa y red discriminativa. Durante la fase de entrenamiento ambas estarán compitiendo unas con otras para obtener un buen resultado.

La red generativa, como su nombre indica, generará muestras a partir de una fuente de datos proporcionada. Por otro lado, la red discriminativa intentará inferir cuáles son imágenes sintetizadas por la red generativa y cuáles provienen de los datos originales proporcionados.

Análogamente, se explica el funcionamiento de las redes generativas adversarias como una competición entre el falsificador y el policía.



A mano izquierda, se visualiza los datos generados por la red generativa y, a mano derecha se observan los datos reales que permiten a la red discriminativa tomar la decisión de si se trata de un dato creado o real. (Fuente: [33])

Figura 4.1: Ejemplo del funcionamiento de una arquitectura GAN

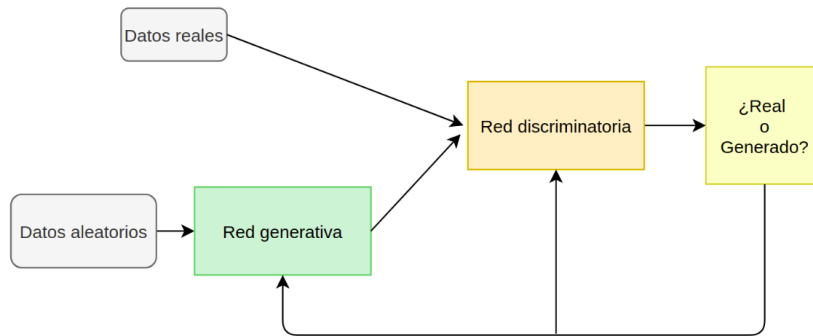
Mientras la tarea del falsificador es crear imitaciones lo más reales posibles, imagínese dinero; el policía intentará encontrar aquellas falsificaciones. Al principio las falsificaciones serán ridículas, muy fáciles de distinguir, pero con el tiempo irán mejorando al punto de no ser capaces de distinguir entre el falso y el real. En esta competición, tanto el falsificador como el policía mejoran su habilidad de generar y discriminar muestras respectivamente. Esto es conocido como juego de suma cero, donde uno gana y el otro pierde.

Por lo cual, dos redes neuronales están interactuando entre sí. Esta arquitectura es conocida como red generativa adversaria, con la que crearemos nuevos datos (nunca antes vistos).

Se analiza a continuación cada uno de estos elementos que componen la red generativa adversaria por separado. En primer lugar, se explica en qué consiste la red generativa y, posteriormente, se desarrolla el concepto de red discriminativa.

### 4.3. Red Generativa

Este elemento de la arquitectura GAN se encarga de generar imágenes para engañar a la red discriminativa. Para su entrenamiento se parte de una entrada de valores aleatorios. Se comienza con unos datos pequeños de entrada y se termina con una salida más grande que la entrada, es decir, se realizan una serie de deconvoluciones en



En esta imagen podemos ver como las redes neuronales interactúan entre sí para dar lugar a la arquitectura GAN

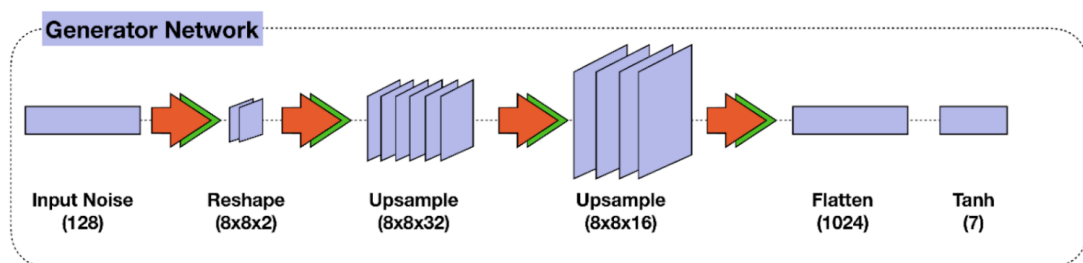
Figura 4.2: Arquitectura Gan

el proceso. Esto se consigue intercalando ceros entre los valores obtenidos y luego se realiza una convolución del mismo.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 2 \\ 0 & 0 & 0 \\ 3 & 0 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 \\ 3 & 4 & 4 \\ 3 & 4 & 4 \end{bmatrix}$$

Un ejemplo sencillo de su funcionamiento es el siguiente: Se parte de una entrada pequeña 2x2. Tras ello, se intercala ceros entre los números originales y se realiza una convolución. Una vez terminado este proceso, se obtiene un entrada de tamaño 3x3.

A continuación se puede ver un ejemplo de arquitectura de la red generativa. Comienza con una entrada aleatoria y realiza tres convoluciones para ampliar los datos de entrada. Si lo se compara con la red discriminativa, uno puede darse cuenta que se realiza justo el proceso contrario: mientras una red busca encontrar características reduciendo la cantidad de datos, la otra red genera características ampliando los datos de salida.



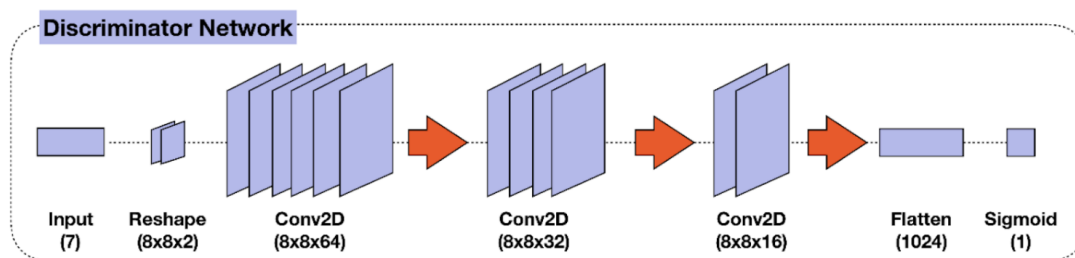
En esta imagen se observa que esta red generativa tiene varias capas. Comienza con una entrada de ruido, luego amplía su tamaño para posteriormente utilizar una tangente hiperbólica como función de activación.

Figura 4.3: Arquitectura de la red generativa. (Fuente: [2])

## 4.4. Red Discriminativa

Este elemento de la arquitectura GAN se encarga de discriminar entre un dato real y otro falso. Se trata de una red neuronal convolucional (convolutional neural network, abreviado CNN). La red convolucional se encargará de encontrar características y patrones de los datos proporcionados, por medio de distintos filtros.

A continuación se observa un ejemplo de una arquitectura de una red neuronal convolucional que actúa de modelo discriminatorio:



En esta imagen se observa que esta red discriminativa comienza con una entrada grande y termina con una entrada pequeña. Entre medias realiza varias convoluciones para obtener características y patrones de los datos.

Figura 4.4: Arquitectura de la red discriminativa. (Fuente: [2])

Una vez visto en qué consiste cada uno de los elementos que componen la arquitectura de una red generativa adversaria, se pasará a analizar los hiper parámetros que tienen las GAN.

## 4.5. Hiper Parámetros

En este apartado se ven cuáles son los hiper parámetros en la arquitectura. Cabe destacar que encontrar la combinación perfecta entre los distintos parámetros que contiene la arquitectura es la parte más difícil, más aún cuando se trata de dos redes neuronales. Ian Goodfellow comenta referente a la búsqueda de la combinación perfecta lo siguiente:

The largest problem facing GANs that researchers should try to resolve is the issue of non-convergence. Most deep models are trained using an optimization algorithm that seeks out a low value of a cost function. While many problems can interfere with optimization, optimization algorithms usually make reliable downhill progress. GANs require finding the equilibrium to a game with two players. Even if each player successfully moves downhill on that player's update, the same update might move the other player uphill. Sometimes the two players eventually reach an equilibrium, but in other scenarios they repeatedly undo each others' progress without arriving anywhere useful.[34]



A nivel general uno puede encontrarse con los siguientes parámetros:

- Tamaño del batch: es el número de muestras escogidas durante el entrenamiento.
- Número de épocas: es el número de pasadas completas sobre los datos de entrenamiento.
- Función de activación: se refiere a la ecuación matemática que determina la salida de la neurona. Algunas funciones son: relu, sigmoide, leaky relu, softmax, tanh, entre otras.
- Algoritmos de optimización: se encarga de actualizar los parámetros de la red neuronal para una mejor predicción. Algunos algoritmos que se utilizan son los siguientes: gradient descent, adagrad, RMSprop or adam, entre otros.
- Operaciones de interpolación: necesarias para obtener los detalles más característicos de una imagen.
- Kernels: tamaño y número de filtros que se deben aplicar al dato.

Además de los anteriores, también se pueden modificar los siguientes parámetros referente a la red discriminativa:

- Tasa de aprendizaje: controla cuán rápido o cuán lento aprende el modelo a predecir.
- Algoritmo de optimización: recomendable utilizar SGD.

Por otro lado, centrándose en los parámetros específicos de la red generativa, se puede modificar lo siguiente:

- Algoritmo de optimización: recomendable utilizar ADAM.
- Función de activación: recomendable utilizar la tangente hiperbólica.



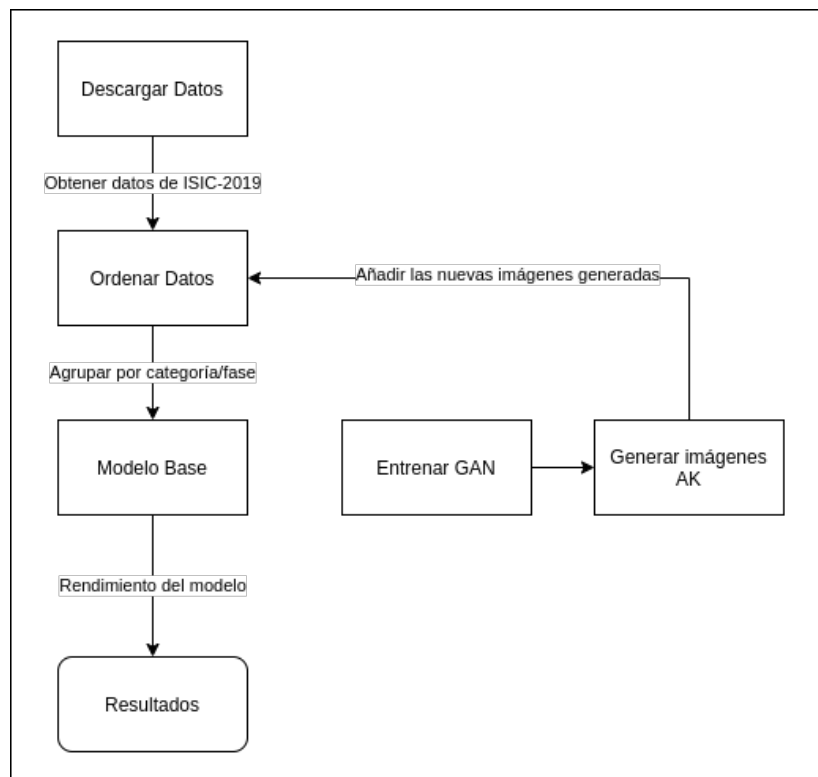
# Capítulo 5

## Creando nuestra GAN

### 5.1. Introducción

Una vez visto en qué consiste, tanto la competición ISIC como la arquitectura generativa adversaria, se pone en práctica los conocimientos adquiridos realizando una serie de pruebas. El propósito de este capítulo es poner en práctica los conocimientos teóricos adquiridos previamente.

Para ello, hemos seguido una serie de pasos que podemos ver en la siguiente figura:



Pasos realizados para crear una red generativa adversaria y probar su funcionamiento.

Figura 5.1: Diagrama de bloque

## 5.2. Crear modelo base

### 5.2.1. Descargar datos

Si uno se dirige a la página de ISIC-2019, <https://challenge2019.isic-archive.com/data.html>, puede encontrar los datos de entrenamiento y los datos de validación. Sin embargo, como los datos de validación no están categorizados, como es de suponer en una competición, solamente se utilizan los datos de entrenamiento.

Tras descargar los datos de entrenamiento, que vienen todos juntos en una carpeta, han de ser organizados. Para ello, se encuentra un csv con los metadatos del conjunto de datos.

Por lo cual, se necesitan dos archivos, `training_input.zip` y `training_metadata.csv`. Tras ser descargados se necesita ejecutar un script para ordenar los datos.

### 5.2.2. Ordenar datos

Para ordenar por categorías cada una de las imágenes, se crea un script que lea la información contenida en el archivo csv y reordene los archivos de `training_input`, generando distintas carpetas representando cada categoría. El script también dividirá el conjunto de datos en 80 % para entrenamiento y 20 % para validación (se puede encontrar el código en la sección de anexos).

### 5.2.3. Crear modelo base

Una vez los datos están ordenados, tanto para la tarea de entrenamiento como para validación, se procede a crear un modelo base. Este modelo base nos servirá como punto para comparar si nuestro modelo basado en la arquitectura GAN mejora el porcentaje de acierto o por el contrario lo empeora.

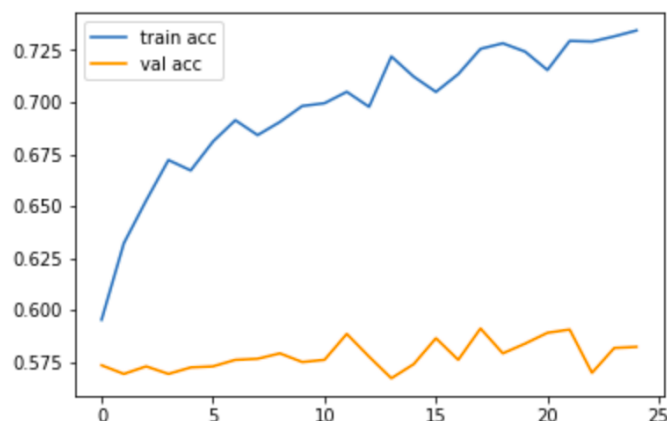
Éste modelo ha sido creado utilizando transfer learning, es decir, se parte de un modelo ya entrenado. El modelo pre-entrenado utilizado es Resnet50, que ha sido entrenado con ImageNet (una base de datos con millones de imágenes, para ser un poco más concretos más de 14 millones).

Para construir este modelo, se utiliza la librería Keras<sup>1</sup> de TensorFlow y se ejecuta en Kaggle,<sup>2</sup> donde se dispone de manera gratuita de GPU's (se puede encontrar el código en la sección de anexos).

Los valores obtenidos son los siguientes:

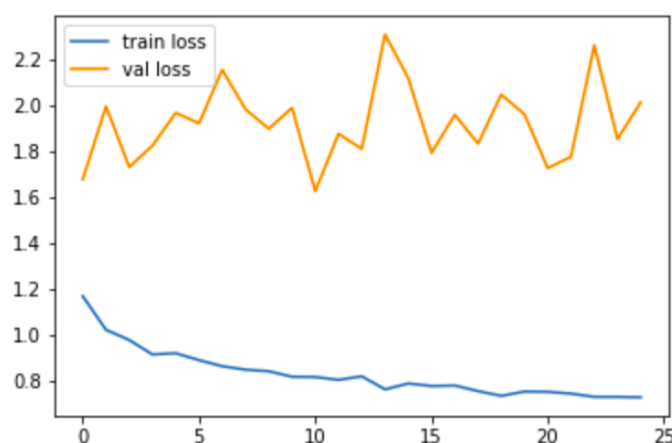
- Fase de entrenamiento: 72,82 % de acierto.
- Fase de validación: 58,23 % de acierto.

Si se grafica los resultados del modelo obtenemos lo siguiente:



Se observa la evolución del acierto del modelo sobre las distintas épocas con la que fue entrenado.

Figura 5.2: Acierto sobre épocas del modelo base



Se observa la evolución de la pérdida del modelo sobre las distintas épocas con la que fue entrenado.

Figura 5.3: Pérdida sobre épocas del modelo base

Observación: dada una serie de problemas que se comentarán más adelante estas gráficas se van a ver modificadas debido a la forma de tratar los datos. Más adelante se comentará el por qué de la decisión de cambio y cómo ésto afectó al modelo base.

## 5.3. Entrenar GAN

Tras finalizar la creación del modelo base, pasamos a crear nuestra red neuronal con una arquitectura generativa adversaria. Creamos tanto la red discriminatoria como la red generativa.

Esta red GAN se entrena con imágenes de cáncer de piel tipo Actinic keratosis (AK) por varias razones:

<sup>1</sup><https://keras.io/applications/#resnet>

<sup>2</sup><https://www.kaggle.com/>

- Tiene el 3,42 % de las imágenes proporcionadas, no es ni el que menos tiene ni el que más.
- El número de imágenes asciende alrededor de 800, siendo unas 650 para la fase de entrenamiento y 150 para la fase de validación.
- Se selecciona únicamente un tipo de cáncer para que el problema sea manejable.

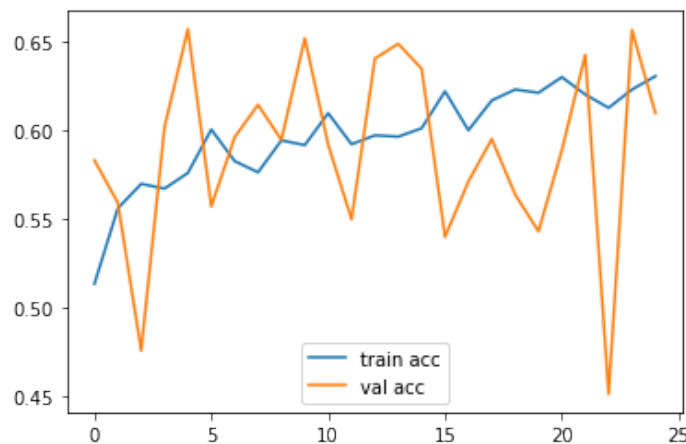
## 5.4. Generar imágenes Actinic Keratosis (AK)

Una vez la red GAN ha sido entrenada, es tiempo de generar nuevas imágenes AK. Sin embargo, aquí es donde se han encontrado varias dificultades:

- El número de imágenes con las que contamos para el entrenamiento de la red GAN es muy reducido.
- El modelo Resnet utilizado para la clasificación de los tipos de cáncer acepta imágenes de tamaño 224x224. Esto conlleva que el proceso de creación de una imagen a ese tamaño requiera mucho tiempo de entrenamiento y muchos recursos.

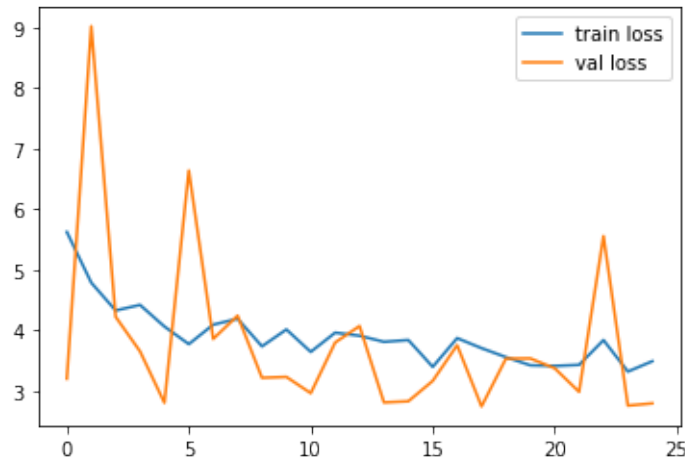
Vistos estos problemas, se decide reducir el tamaño de las imágenes a 64x64 para que el proceso de generación fuese menos complejo y costoso. Sin embargo, esto implicaba que el modelo base creado debía ser modificado. Todas las imágenes deberían ser de tamaño 64x64 y el modelo ser entrenado con éstas.

A continuación se puede ver el resultado de la ejecución siguiendo esta premisa:



Se observa la evolución del acierto del modelo sobre las distintas épocas con la que fue entrenado. En la fase de entrenamiento se puede observar que progresivamente va aprendiendo y llega a un 62 % de acierto. Sin embargo, en la fase de evaluación el porcentaje de acierto varía desde el 45 % al 65 % dependiendo de la época, reflejando que el modelo no ha sido capaz de generalizar suficientemente bien.

Figura 5.4: Acierto sobre épocas del modelo base (tamaño imagen 64x64)



Se observa la evolución de la pérdida del modelo sobre las distintas épocas con la que fue entrenado. Se obtiene un comportamiento similar al reflejado en la figura anterior sobre el acierto.

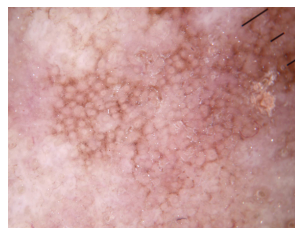
Figura 5.5: Pérdida sobre épocas del modelo base (tamaño imagen 64x64)

Una vez comentada esta dificultad, y cómo se ha buscado solucionarla, se trata a continuación los resultados obtenidos al generar imágenes de tipo Actinic Keratosis. Se decide crear un 10 % de imágenes del tipo AK, es decir 80 imágenes, ya que el conjunto de datos que se tenía era de 800 imágenes aproximadamente (como en las secciones anteriores, el código, tanto para entrenar la GAN como para generar imágenes, se encuentre en la sección de anexos).

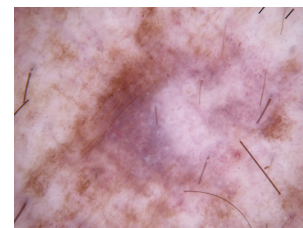
Abajo se encuentran distintas imágenes reales del tipo de cáncer de piel Actinic Keratosis, todas han sido sacadas de las imágenes proporcionadas por el ISIC-2019.



(a) ejemplo 1



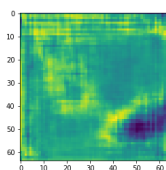
(b) ejemplo 2



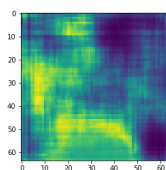
(c) ejemplo 3

Figura 5.6: Imágenes del tipo cáncer AK

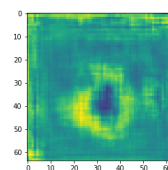
Éstas son las imágenes que nuestra red ha sido capaz de crear.



(a) generado 1



(b) generado 2



(c) generado 3

Figura 5.7: Imágenes generadas por la red GAN

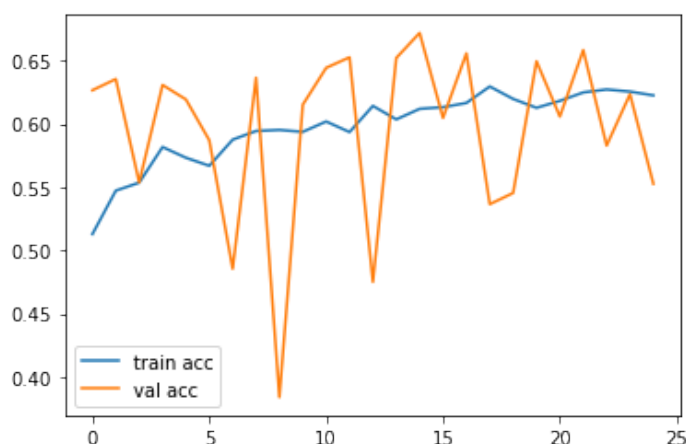
A primera vista, las imágenes generadas no son nada parecidas a las imágenes originales. Sin embargo, se observa que la red ha sido capaz de generar imágenes y de generar distintos patrones. Si acertase en el color, algunos de las imágenes generadas podrían asemejarse a las imágenes proporcionadas.

## 5.5. Incluir las imágenes generadas en el conjunto de datos original

Una vez las 80 imágenes se generan, es tiempo de incluir éstas en el conjunto de datos original, proporcionado por la ISIC-2019. Aquí se toma la decisión de estas 80 imágenes incluirlas exclusivamente en la parte de entrenamiento, ya que el objetivo es ver si el modelo base aprende mejor con la ayuda de imágenes sintéticas.

## 5.6. Modelo base con las imágenes generadas

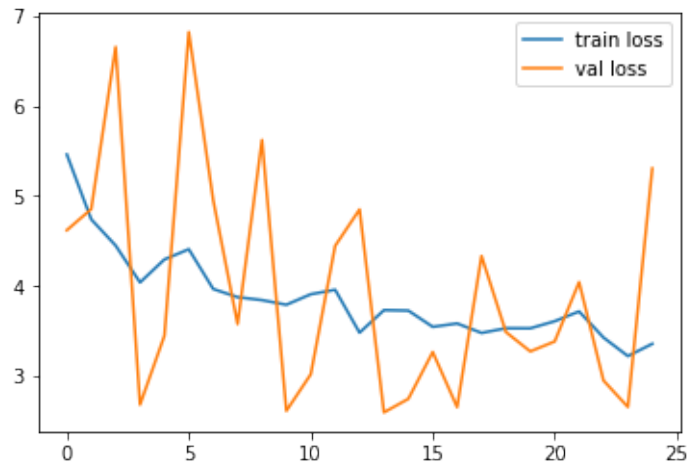
Tras incluir las 80 imágenes en el conjunto de datos original se vuelve a entrenar y validar el modelo base. A continuación, se observa los resultados obtenidos tras añadir las imágenes generadas por la red GAN.



Se ve la evolución del acierto del modelo sobre las distintas épocas con la que fue entrenado. En la fase de entrenamiento se puede observar que progresivamente va aprendiendo y llega a un 61 % de acierto (muy similar al valor obtenido si se entrena sin las imágenes nuevas del tipo AK). Sin embargo, en la fase de validación el porcentaje de acierto varía desde el 40 % al 66 % dependiendo de la época, reflejando que el modelo no ha sido capaz de generalizar suficientemente bien (el comportamiento es muy similar al obtenido sin las nuevas imágenes AK).

Figura 5.8: Acierto sobre épocas del modelo base con imágenes AK generadas





Se ve la evolución de la pérdida del modelo sobre las distintas épocas con la que fue entrenado. Obtenemos un comportamiento similar al reflejado en la figura anterior sobre el acierto. Sin embargo, aquí se observa que el modelo es peor que el entrenado sin imágenes generadas, ya que la variación en la época de validación es significativamente mayor que en el modelo anterior.

Figura 5.9: Pérdida sobre épocas del modelo base con imágenes AK generadas

En el siguiente capítulo se habla sobre las conclusiones extraídas tras realizar este experimento y algunas observaciones sobre cómo mejorar el proyecto, en una futura fase.



# Capítulo 6

## Conclusiones y trabajo futuro

### 6.1. Conclusiones

El objetivo principal de este Trabajo Fin de Grado ha sido el poder adentrarse en la arquitectura GAN. Para ello, primero se vio en qué consiste y cuáles son sus características. Tras ello, se ha podido aprender sobre la competición ISIC, donde cada año distintas instituciones o particulares participan para crear modelos capaces de clasificar los distintos tipos de cáncer de piel que existen.

Una vez adquirido este conocimiento, se ha puesto en práctica la teoría y se ha generado 80 imágenes, en específico del tipo Actinic Keratosis. Todo esto con el objetivo de ver si se podría aplicar una red GAN como data augmentation. Sin embargo, los resultados obtenidos no han sido muy buenos: las imágenes generadas se parecían poco a las originales, aunque tenía ciertos rasgos.

En el capítulo anterior se vio cómo se comportaba el modelo base, sin y con las nuevas imágenes generadas. Se observó que su rendimiento era muy similar, aunque con las nuevas imágenes era un poco peor. En mi opinión, la similitud del rendimiento se debía a la inclusión de únicamente 80 imágenes. Si éstas hubieran sido 800, por ejemplo, el rendimiento hubiese sido mucho peor.

También se realizó una comparativa para ver si predecía mejor la categoría con las nuevas imágenes, no comparando si el modelo en su conjunto era mejor, sólo si prediciendo el tipo AK era mejor o no. El resultado es sorprendente porque sí mejora en unas décimas.

Classification Report				
	precision	recall	f1-score	support
AK	0.00	0.00	0.00	173
ECC	0.13	0.16	0.14	664
EKL	0.09	0.12	0.10	524
DF	0.02	0.06	0.03	47
MFI	0.20	0.18	0.19	904
NV	0.50	0.45	0.48	2575
SCC	0.02	0.05	0.03	125
VASC	0.00	0.00	0.00	50
accuracy			0.30	5062
macro avg	0.12	0.13	0.12	5062
weighted avg	0.32	0.30	0.31	5062

(a) Modelo base solo con las imágenes proporcionadas

Classification Report				
	precision	recall	f1-score	support
AK	0.06	0.01	0.01	173
BCC	0.14	0.28	0.19	664
BKL	0.10	0.03	0.05	524
DF	0.00	0.00	0.00	47
MEL	0.18	0.40	0.25	904
NV	0.53	0.31	0.39	2575
SCC	0.00	0.00	0.00	125
VASC	0.00	0.00	0.00	50
accuracy			0.27	5062
macro avg	0.13	0.13	0.11	5062
weighted avg	0.33	0.27	0.27	5062

(b) Modelo base con nuevas imágenes AK

Figura 6.1: Modelo base con los datos proporcionados por ISIC-2019 (izquierda) y Modelo base con imágenes AK generadas

Algunas de las dificultades, y que han condicionado los resultados, son los siguientes:

- La distribución de los datos proporcionados por ISIC-2019 es muy dispar. Muchos de los tipos de cáncer tenían menos del 4% de representación en la muestra, dificultando en gran manera su clasificación.
- El tipo de cáncer que se eligió sólo constaba de alrededor de 800 imágenes, por lo que había una carencia de datos importante, si hablamos de redes neuronales.
- Hubo un problema de tamaño de la imagen, ya que se buscaba en un principio imágenes de tamaño 224x224, pero su tiempo de computación era muy elevado. En la plataforma donde se realizaron las pruebas únicamente se permitían 30 horas a la semana con intervalos de no más de 9 horas seguidas de ejecución, teniendo que modificar el notebook cada hora para que no se cerrase la sesión, por lo que se hacía muy complicado dejar entrenando por mucho tiempo la red GAN creada.

Aún a pesar de estas dificultades encontradas, he podido ir aprendiendo mientras este proyecto se llevaba a cabo. Es muy intrigante lo que se llegará a ser capaz de realizar en un futuro cercano. Algunos pensamientos que saco tras realizar este trabajo de fin de grado son los siguientes:

- Es muy interesante la idea de poder generar nuevas imágenes de manera sintética. Las aplicaciones son grandes, como se vio en los capítulos anteriores.
- Aunque hay algunas arquitecturas GAN que trabajan con pocos datos, sigue siendo un punto de inflexión. Con esto no me refiero a tener millones de datos, pero sí una buena cantidad de datos de calidad.
- Creo que una red GAN podría ser muy útil para realizar data augmentation, aunque tiene que haber una cantidad de datos importante (o crear modelos capaces de dar buenos resultados con muy pocos datos). Es muy difícil trabajar con pocos datos.

## 6.2. Trabajo futuro

A la vista de los resultados que se han obtenido, me parecería interesante ahondar en lo siguiente en un futuro:

- ¿Se puede aplicar el data augmentation con la arquitectura GAN en la mayoría de los problemas?
- ¿Cómo se puede afrontar un conjunto de datos que está muy desbalanceado, es decir, donde algunas categorías tienen mucho peso mientras que otras muy poco?  
¿Puede una red GAN ser pieza diferenciante para este tipo de problemas?



# Bibliografía

- [1] D. Bisla, A. Choromanska, J. A. Stein, D. Polsky, and R. Berman, “Towards Automated Melanoma Detection with Deep Learning: Data Purification and Augmentation,” *arXiv e-prints*, p. arXiv:1902.06061, Feb. 2019. [ix](#), [13](#), [14](#), [15](#), [16](#)
- [2] R. Di Sipio, M. F. Giannelli, S. K. Haghighat, and S. Palazzo, “DijetGAN: a Generative-Adversarial Network approach for the simulation of QCD dijet events at the LHC,” *Journal of High Energy Physics*, vol. 2019, p. 110, Aug. 2019. [ix](#), [ix](#), [23](#), [24](#)
- [3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv e-prints*, p. arXiv:1512.03385, Dec. 2015. [3](#)
- [4] C. Baskin, N. Liss, A. Mendelson, and E. Zheltonozhskii, “Streaming architecture for large-scale quantized neural networks on an fpga-based dataflow platform,” 07 2017. [4](#)
- [5] L. G. Solutions, “Apply laplacian filters.” [4](#)
- [6] missinglink.ai, “Fully connected layers in convolutional neural networks: The complete guide.” [6](#)
- [7] T. L. I. Sugata and C. K. Yang, “Leaf App: Leaf recognition with deep convolutional neural networks,” vol. 273, p. 012004, Nov. 2017. [6](#)
- [8] T. Karras, T. Aila, S. Laine, and J. Lehtinen, “Progressive Growing of GANs for Improved Quality, Stability, and Variation,” *arXiv e-prints*, p. arXiv:1710.10196, Oct. 2017. [7](#)
- [9] P. SHARMA, “What are generative models and gans? the magic of computer vision.” [7](#)
- [10] A. Radford, L. Metz, and S. Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,” *arXiv e-prints*, p. arXiv:1511.06434, Nov. 2015. [8](#)
- [11] T. Karras, T. Aila, S. Laine, and J. Lehtinen, “Progressive Growing of GANs for Improved Quality, Stability, and Variation,” *arXiv e-prints*, p. arXiv:1710.10196, Oct. 2017. [8](#)
- [12] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-Image Translation with Conditional Adversarial Networks,” *arXiv e-prints*, p. arXiv:1611.07004, Nov. 2016. [9](#)

- [13] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, “Generative Adversarial Text to Image Synthesis,” *arXiv e-prints*, p. arXiv:1605.05396, May 2016. 9
- [14] M. E. C. B. H. M. A. M. S. W. D. A. K. K. L. N. M. H. K. A. H. Noel C. F. Codella, David Gutman, “Skin lesion analysis toward melanoma detection: A challenge at the 2017 international symposium on biomedical imaging (isbi), hosted by the international skin imaging collaboration (isic),” 2017. 11
- [15] R. C. . K. H. Tschandl P., “The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions,” 2018. 11
- [16] V. R. B. H. V. V. O. R. A. C. H. S. P. J. M. Marc Combalia, Noel C. F. Codella, “Bcn20000: Dermoscopic lesions in the wild,” 2019. 11
- [17] M. S. R. W. Nils Gessert, Maximilian Nielsen and A. Schlaefer, “Skin Lesion Classification Using Loss Balancing and Ensembles of Multi-Resolution Efficient-Nets,” *Institute of Medical Technology, Hamburg University of Technology, Hamburg, Germany*, 2019. 12
- [18] R. M. Steven Zhou, Yixin Zhuang, “Multi-Category Skin Lesion Diagnosis Using Dermoscopy Images and Deep CNN Ensembles,” *Dysion AI Technology Co*, 2019. 12
- [19] M. P. F. B. R. P. C. G. A. A. Federico Pollastri, Juan Maroñas, “AImageLab-PRHLT at ISIC Challenge 2019,” *Universit’a degli Studi di Modena e Reggio Emilia, Italy Universitat Politecnica de Valencia, Valencia*, 2019. 12
- [20] A.-R. A. Andre G. C. Pacheco and T. Trappenberg, “Skin cancer detection based on deep learning and entropy to detect outlier samples,” *Graduate Program in Computer Science, Federal University of Espirito Santo, Brazil; Faculty of Computer Science, Dalhousie University, Canada; Faculty of Natural Sciences, Computing Science and Mathematics, University of Stirling, United Kingdom*, 2019. 12
- [21] V. Chouhan, “Skin Lesion Analysis towards Melanoma Detection with Deep Convolutional Neural Network,” *nurithmlabs.tech*), 2019. 12
- [22] J. Y. Aleksey Nozdryn-Plotnicki and W. Yolland, “Ensembling Convolutional Neural Networks for Skin Cancer Classification,” *Jordan Yap*, 2019. 12
- [23] F. M. R. S. H. K. I. B. R. W. Nils Gessert, Thilo Sentker and A. Schlaefer, “Skin Lesion Diagnosis using Ensembles, Unscaled Multi-Crop Evaluation and Loss Weighting,” *DAISYlab, Forschungszentrum Medizintechnik Hamburg, Germany*, 2019. 12
- [24] S. M. R. W. J. Z. J. L. J. P. G. J. Z. Y. Jiaxin Zhuang, Weipeng Li, “Skin Lesion Analysis Towards Melanoma Detection Using Deep Neural Network Ensemble,” *MIA Group, School of Data and Computer Science, Sun Yet-sen University, China Computing School of Science and Engineering, University of Dundee. Department of computer science, University of Jaffna*, 2019. 12



- [25] K. Matsunaga, A. Hamada, A. Minagawa, and H. Koga, “Image Classification of Melanoma, Nevus and Seborrheic Keratosis by Deep Neural Network Ensemble,” *arXiv e-prints*, p. arXiv:1703.03108, Mar. 2017. 12
- [26] I.ález Díaz@, “Incorporating the Knowledge of Dermatologists to Convolutional Neural Networks for the Diagnosis of Skin Lesions,” *arXiv e-prints*, p. arXiv:1703.01976, Mar. 2017. 12
- [27] A. Menegola, J. Tavares, M. Fornaciali, L. Tzy Li, S. Avila, and E. Valle, “RECOD Titans at ISIC Challenge 2017,” *arXiv e-prints*, p. arXiv:1703.04819, Mar. 2017. 12
- [28] L. Bi, J. Kim, E. Ahn, and D. Feng, “Automatic Skin Lesion Analysis using Large-scale Dermoscopy Images and Deep Residual Networks,” *arXiv e-prints*, p. arXiv:1703.04197, Mar. 2017. 12
- [29] X. Yang, Z. Zeng, S. Y. Yeo, C. Tan, H. L. Tey, and Y. Su, “A Novel Multi-task Deep Learning Model for Skin Lesion Segmentation and Classification,” *arXiv e-prints*, p. arXiv:1703.01025, Mar. 2017. 12
- [30] I. Saad Ali, M. Farouk Mohamed, and Y. Bassyouni Mahdy, “Data Augmentation for Skin Lesion using Self-Attention based Progressive Generative Adversarial Network,” *arXiv e-prints*, p. arXiv:1910.11960, Oct. 2019. 13, 17, 18
- [31] S. Ding, “Skin Lesion Analysis Towards Melanoma Detection Using Generative Adversarial Network,” *Isic2019*, 2019. 13, 18
- [32] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, “High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs,” *arXiv e-prints*, p. arXiv:1711.11585, Nov. 2017. 18
- [33] G. Developers, “Overview of gan structure.” 22
- [34] I. Goodfellow, “NIPS 2016 Tutorial: Generative Adversarial Networks,” *arXiv e-prints*, p. 34, Dec. 2016. 24
- [35] J. Heaton, “T81-558: Applications of deep neural networks.” 52



# Apéndice



# Apéndice A

## Códigos

En este anexo se pueden encontrar los códigos utilizados para la implementación del modelo base, red GAN y preprocesado de los datos.

### A.1. Ordenar Datos

A continuación se puede ver el código utilizado para realizar la ordenación de los datos descargados de la página oficial de la competición ISIC-2019. Una vez descargados, se organizan en los distintos tipos de cáncer y se separa las imágenes que se utilizarán tanto para la fase de entrenamiento como de evaluación.

---

```
1 import pandas as pd
2 import numpy as np
3 import os
4 import shutil
5 import math
6
7 def move_file(file, src, dest):
8     src_path = os.path.join(src, file)
9     dest_path = os.path.join(dest, file)
10    shutil.move(src_path, dest_path)
11
12
13 def create_folder(path, name):
14     folder_name = os.path.join(path, name)
15     os.mkdir(folder_name)
16
17
18 # split data into train and test - return list of file names
19 def split_files(Arr, name, percentage=0.8):
20     rows = Arr.shape[0]
21     indices = np.arange(rows)
22     permuted = np.random.permutation(indices)
23     p = math.ceil(rows*percentage)
24     return Arr[permuted[:p],], Arr[permuted[p:],]
25
26
27 def fill_folder(folder_name, files):
28     for file in files:
29         move_file(file+'.jpg', data_ISIC, folder_name)
30
31
32 # set paths
33 work_path = os.path.abspath(os.getcwd())
34 data_ISIC = os.path.join(work_path, 'ISIC_2019_Training_Input')
35 ground_truth_file = os.path.join(work_path, 'ISIC_2019_Training_GroundTruth.csv')
36
37
38 # create data folder and inside train and test folders
39 create_folder(work_path, 'data')
40 data_path = os.path.join(work_path, 'data')
41
42 create_folder(data_path, 'train')
43 data_train_path = os.path.join(data_path, 'train')
44
45 create_folder(data_path, 'test')
46 data_test_path = os.path.join(data_path, 'test')
47
48
```

```

49 # load information about the dataset
50 df = pd.read_csv(ground_truth_file)
51
52 # remove unnecessary column
53 df = df.iloc[:, :-1]
54
55 # function to get data easily
56 get_data = lambda df, col: df[df[column] == 1].iloc[:, 0].to_numpy()
57
58 # iterate over all cancer types - avoid column filename (image)
59 for_train = for_test = 0
60 for column in df.columns[1:]:
61     train, test = split_files(get_data(df, column), column)
62
63     for_train, for_test = for_train + len(train), for_test + len(test)
64     print(f'Train: {len(train)} - Test {len(test)}')
65
66     create_folder(data_train_path, column)
67     fill_folder(os.path.join(data_train_path, column), train)
68
69     create_folder(data_test_path, column)
70     fill_folder(os.path.join(data_test_path, column), test)
71
72 print(f'Nº Train data: {for_train}\nNº Test data: {for_test}')
73
74 # remove folder ISIC
75 shutil.rmtree(data_ISIC)

```

---

Primero, se realiza las importaciones de las librerías necesarias. Pandas y Numpy para trabajar con dataframes. Os y Shutil para interactuar con el sistema. Math para utilizar la función suelo.

```

1 import pandas as pd
2 import numpy as np
3 import os
4 import shutil
5 import math

```

Se crea distintas funciones necesarias para mover un archivo, otra para mover un conjunto de archivos, también para crear una carpeta y para dividir los datos (unos irán para la fase de entrenamiento y otros para la fase de validación).

```

7 def move_file(file, src, dest):
8     src_path = os.path.join(src, file)
9     dest_path = os.path.join(dest, file)
10    shutil.move(src_path, dest_path)
11
12
13 def create_folder(path, name):

```

```

14     folder_name = os.path.join(path, name)
15     os.mkdir(folder_name)
16
17
18     # split data into train and test - return list of file names
19     def split_files(Arr, name, percentage=0.8):
20         rows = Arr.shape[0]
21         indices = np.arange(rows)
22         permuted = np.random.permutation(indices)
23         p = math.ceil(rows*percentage)
24         return Arr[permuted[:p],], Arr[permuted[p:],]
25
26
27     def fill_folder(folder_name, files):
28         for file in files:
29             move_file(file+'.jpg', data_ISIC, folder_name)

```

Establecemos las rutas relativas y creamos las carpetas de data, train y test.

```

32     # set paths
33     work_path = os.path.abspath(os.getcwd())
34     data_ISIC = os.path.join(work_path, 'ISIC_2019_Training_Input')
35     ground_truth_file = os.path.join(work_path, 'ISIC_2019_Training_GroundTruth.csv')
36
37
38     # create data folder and inside train and test folders
39     create_folder(work_path, 'data')
40     data_path = os.path.join(work_path, 'data')
41
42     create_folder(data_path, 'train')
43     data_train_path = os.path.join(data_path, 'train')
44
45     create_folder(data_path, 'test')
46     data_test_path = os.path.join(data_path, 'test')

```

Se leen los metadatos, cargando el archivo csv en un dataframe y eliminamos una columna innecesaria. También se crea una función para obtener los datos de una columna específica.

```

49     # load information about the dataset
50     df = pd.read_csv(ground_truth_file)
51
52     # remove unnecessary column
53     df = df.iloc[:, :-1]
54
55     # function to get data easily
56     get_data = lambda df, col: df[df[column] == 1].iloc[:, 0].to_numpy()

```



Se itera sobre cada tipo de cáncer. Se realiza la división de datos que irán para entrenamiento y aquellos que irán para validación. Posteriormente, se crea la carpeta con el nombre del tipo de cáncer específico y se rellena con las imágenes pertinentes. Se termina eliminando la carpeta original, ya que ahora no contiene nada (todo está ubicado en 'data', dentro de su respectiva categoría).

```

58 # iterate over all cancer types - avoid column filename (image)
59 for_train = for_test = 0
60 for column in df.columns[1:]:
61     train, test = split_files(get_data(df, column), column)
62
63     for_train, for_test = for_train + len(train), for_test + len(test)
64     print(f'Train: {len(train)} - Test {len(test)}')
65
66     create_folder(data_train_path, column)
67     fill_folder(os.path.join(data_train_path, column), train)
68
69     create_folder(data_test_path, column)
70     fill_folder(os.path.join(data_test_path, column), test)
71
72 print(f'Nº Train data: {for_train}\nNº Test data: {for_test}')
73
74 # remove folder ISIC
75 shutil.rmtree(data_ISIC)

```

## A.2. Modelo Base

A continuación se encuentra el código necesario para crear nuestro modelo base (un clasificador de imágenes), que servirá para verificar si utilizando una red GAN para generar imágenes mejora el proceso de clasificación.

---

```

1 import os
2 data_path = '../input/isic2019/data'
3
4 from tensorflow.python.keras.applications.resnet import ResNet50
5 from tensorflow.python.keras.models import Sequential
6 from tensorflow.python.keras.layers import Dense, Flatten, GlobalAveragePooling2D
7 from tensorflow.python.keras.applications.resnet import preprocess_input
8 from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
9 import matplotlib.pyplot as plt
10 %matplotlib inline
11
12 num_classes = 8
13
14 my_model = Sequential()
15 my_model.add(ResNet50(include_top=False, pooling='avg', weights='imagenet'))

```

```
16 my_model.add(Dense(num_classes, activation='softmax'))
17
18 # Say not to train first layer (ResNet) model. It is already trained
19 my_model.layers[0].trainable = False
20
21 my_model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
22
23 image_size = 224
24 data_generator = ImageDataGenerator(preprocessing_function=preprocess_input)
25
26 train_generator = data_generator.flow_from_directory(
27     os.path.join(data_path, 'train'),
28     target_size=(image_size, image_size),
29     batch_size=24,
30     class_mode='categorical')
31
32 validation_generator = data_generator.flow_from_directory(
33     os.path.join(data_path, 'test'),
34     target_size=(image_size, image_size),
35     class_mode='categorical')
36
37 # fit the model
38 r = my_model.fit_generator(
39     train_generator,
40     steps_per_epoch=250,
41     validation_steps=60,
42     validation_data=validation_generator,
43     epochs=25
44 )
45
46 # loss
47 plt.plot(r.history['loss'], label='train loss')
48 plt.plot(r.history['val_loss'], label='val loss')
49 plt.legend()
50 plt.show()
51 plt.savefig('base_model_LossVal_loss')
52
53 # accuracies
54 plt.plot(r.history['accuracy'], label='train acc')
55 plt.plot(r.history['val_accuracy'], label='val acc')
56 plt.legend()
57 plt.show()
58 plt.savefig('base_model_AccVal_acc')
59
60 import tensorflow as tf
61
62 from keras.models import load_model
63
```

```
64 my_model.save('base_model.h5')
```

---

Se realiza una serie de importaciones que necesitaremos más adelante.

```
1  import os
2  data_path = '../input/isic2019/data'
3
4  from tensorflow.python.keras.applications.resnet import ResNet50
5  from tensorflow.python.keras.models import Sequential
6  from tensorflow.python.keras.layers import Dense, Flatten, GlobalAveragePooling2D
7  from tensorflow.python.keras.applications.resnet import preprocess_input
8  from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
9  import matplotlib.pyplot as plt
10 %matplotlib inline
```

Se define el número de categorías que clasificará el modelo. Se importa el modelo pre-entrenado ResNet50, con un pooling de promedio y los pesos originales de ImageNet. Posteriormente, se añade una nueva capa con el número de categorías como neuronas. Por lo cual, la red tendrá dos capas, la primera es el modelo ResNet50 y, la segunda capa, contiene ocho neuronas. Este modelo tiene que entrenar esta segunda capa para que pueda predecir acorte a este problema de clasificación.

```
12 num_classes = 8
13
14 my_model = Sequential()
15 my_model.add(ResNet50(include_top=False, pooling='avg', weights='imagenet'))
16 my_model.add(Dense(num_classes, activation='softmax'))
17
18 # Say not to train first layer (ResNet) model. It is already trained
19 my_model.layers[0].trainable = False
```

Posteriormente, se compila el modelo y se cargan las imágenes, tanto para la fase de entrenamiento como para la de validación. Se utiliza un tamaño de batch de 24.

```
21 my_model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
22
23 image_size = 224
24 data_generator = ImageDataGenerator(preprocessing_function=preprocess_input)
25
26 train_generator = data_generator.flow_from_directory(
27     os.path.join(data_path, 'train'),
28     target_size=(image_size, image_size),
29     batch_size=24,
30     class_mode='categorical')
31
32 validation_generator = data_generator.flow_from_directory(
33     os.path.join(data_path, 'test'),
34     target_size=(image_size, image_size),
35     class_mode='categorical')
```

Es momento de entrenar el modelo. Se realiza con 25 épocas, 60 pasos por validación y 250 pasos por época.

```

38 # fit the model
39 r = my_model.fit_generator(
40     train_generator,
41     steps_per_epoch=250,
42     validation_steps=60,
43     validation_data=validation_generator,
44     epochs=25
45 )

```

Se grafican los resultados obtenidos y se salva el modelo entrenado.

```

46 # loss
47 plt.plot(r.history['loss'], label='train loss')
48 plt.plot(r.history['val_loss'], label='val loss')
49 plt.legend()
50 plt.show()
51 plt.savefig('base_model_LossVal_loss')
52
53 # accuracies
54 plt.plot(r.history['accuracy'], label='train acc')
55 plt.plot(r.history['val_accuracy'], label='val acc')
56 plt.legend()
57 plt.show()
58 plt.savefig('base_model_AccVal_acc')
59
60 import tensorflow as tf
61
62 from keras.models import load_model
63
64 my_model.save('base_model.h5')

```

### A.3. Red GAN

Me he basado en el código de Jeff Heaton[35]/.

---

```

1 import tensorflow as tf
2 from tensorflow.keras.layers import Input, Reshape, Dropout, Dense, Flatten, BatchNormalization, Act
3 from tensorflow.keras.layers import LeakyReLU
4 from tensorflow.keras.layers import UpSampling2D, Conv2D
5 from tensorflow.keras.models import Sequential, Model, load_model
6 from tensorflow.keras.optimizers import Adam
7 import numpy as np
8 from PIL import Image

```

```

9  from tqdm import tqdm
10 import os
11 import time
12 import matplotlib.pyplot as plt
13
14 # Generation resolution - Must be square
15 # Training data is also scaled to this.
16 # Note GENERATE_RES 4 or higher will blow Google CoLab's memory and have not
17 # been tested extensively.
18 GENERATE_RES = 2 # Generation resolution factor (1=32, 2=64, 3=96, 4=128, etc.)
19 GENERATE_SQUARE = 32 * GENERATE_RES # rows/cols (should be square)
20 IMAGE_CHANNELS = 3
21
22 # Preview image
23 PREVIEW_ROWS = 7
24 PREVIEW_COLS = 7
25 PREVIEW_MARGIN = 16
26
27 # Size vector to generate images from
28 SEED_SIZE = 100
29
30 # Configuration
31 DATA_PATH_1 = '../input/isic2019/data/train/AK'
32 DATA_PATH_2 = '../input/isic2019/data/test/AK'
33 EPOCHS = 1000
34 BATCH_SIZE = 32
35 BUFFER_SIZE = 60000
36
37 print(f"Will generate {GENERATE_SQUARE}px square images.")
38
39 # Nicely formatted time string
40 def hms_string(sec_elapsed):
41     h = int(sec_elapsed / (60 * 60))
42     m = int((sec_elapsed % (60 * 60)) / 60)
43     s = sec_elapsed % 60
44     return "{:}>02}{:}>05.2f)".format(h, m, s)
45 # Image set has 11,682 images. Can take over an hour for initial preprocessing.
46 # Because of this time needed, save a Numpy preprocessed file.
47 # Note, that file is large enough to cause problems for some versions of Pickle,
48 # so Numpy binary files are used.
49 training_binary_path = os.path.join('/kaggle/working', f'training_data_{GENERATE_SQUARE}_{GENERATE_RES}')
50
51 print(f"Looking for file: {training_binary_path}")
52
53 if not os.path.isfile(training_binary_path):
54     start = time.time()
55     print("Loading training images...")
56

```

```

57     training_data = []
58     for ak_images in [DATA_PATH_1, DATA_PATH_2]:
59
60         #ak_images = os.path.join(DATA_PATH, 'ak_images')
61
62         for filename in tqdm(os.listdir(ak_images)):
63             path = os.path.join(ak_images, filename)
64             image = Image.open(path).resize((GENERATE_SQUARE, GENERATE_SQUARE), Image.ANTIALIAS)
65             training_data.append(np.asarray(image))
66     training_data = np.reshape(training_data, (-1, GENERATE_SQUARE, GENERATE_SQUARE, IMAGE_CHANNELS))
67     training_data = training_data.astype(np.float32)
68     training_data = training_data / 127.5 - 1.
69
70
71     print("Saving training image binary...")
72     np.save(training_binary_path, training_data)
73     elapsed = time.time() - start
74     print(f'Image preprocess time: {hms_string(elapsed)}')
75 else:
76     print("Loading previous training pickle...")
77     training_data = np.load(training_binary_path)
78
79     # Batch and shuffle the data
80     train_dataset = tf.data.Dataset.from_tensor_slices(training_data).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
81
82     def build_generator(seed_size, channels):
83         model = Sequential()
84
85         model.add(Dense(4*4*256, activation="relu", input_dim=seed_size))
86         model.add(Reshape((4, 4, 256)))
87
88         model.add(UpSampling2D())
89         model.add(Conv2D(256, kernel_size=3, padding="same"))
90         model.add(BatchNormalization(momentum=0.8))
91         model.add(Activation("relu"))
92
93         model.add(UpSampling2D())
94         model.add(Conv2D(256, kernel_size=3, padding="same"))
95         model.add(BatchNormalization(momentum=0.8))
96         model.add(Activation("relu"))
97
98         # Output resolution, additional upsampling
99         model.add(UpSampling2D())
100        model.add(Conv2D(128, kernel_size=3, padding="same"))
101        model.add(BatchNormalization(momentum=0.8))
102        model.add(Activation("relu"))
103
104        if GENERATE_RES > 1:

```

```

105         model.add(UpSampling2D(size=(GENERATE_RES,GENERATE_RES)))
106         model.add(Conv2D(128,kernel_size=3,padding="same"))
107         model.add(BatchNormalization(momentum=0.8))
108         model.add(Activation("relu"))
109
110     # Final CNN layer
111     model.add(Conv2D(channels,kernel_size=3,padding="same"))
112     model.add(Activation("tanh"))
113
114     return model
115
116
117 def build_discriminator(image_shape):
118     model = Sequential()
119
120     model.add(Conv2D(32, kernel_size=3, strides=2, input_shape=image_shape, padding="same"))
121     model.add(LeakyReLU(alpha=0.2))
122
123     model.add(Dropout(0.25))
124     model.add(Conv2D(64, kernel_size=3, strides=2, padding="same"))
125     model.add(ZeroPadding2D(padding=((0,1),(0,1))))
126     model.add(BatchNormalization(momentum=0.8))
127     model.add(LeakyReLU(alpha=0.2))
128
129     model.add(Dropout(0.25))
130     model.add(Conv2D(128, kernel_size=3, strides=2, padding="same"))
131     model.add(BatchNormalization(momentum=0.8))
132     model.add(LeakyReLU(alpha=0.2))
133
134     model.add(Dropout(0.25))
135     model.add(Conv2D(256, kernel_size=3, strides=1, padding="same"))
136     model.add(BatchNormalization(momentum=0.8))
137     model.add(LeakyReLU(alpha=0.2))
138
139     model.add(Dropout(0.25))
140     model.add(Conv2D(512, kernel_size=3, strides=1, padding="same"))
141     model.add(BatchNormalization(momentum=0.8))
142     model.add(LeakyReLU(alpha=0.2))
143
144     model.add(Dropout(0.25))
145     model.add(Flatten())
146     model.add(Dense(1, activation='sigmoid'))
147
148     return model
149
150 def save_images(cnt,noise):
151     image_array = np.full((
152         PREVIEW_MARGIN + (PREVIEW_ROWS * (GENERATE_SQUARE+PREVIEW_MARGIN)),

```

```

153     PREVIEW_MARGIN + (PREVIEW_COLS * (GENERATE_SQUARE+PREVIEW_MARGIN)), 3),
154     255, dtype=np.uint8)
155
156     generated_images = generator.predict(noise)
157
158     generated_images = 0.5 * generated_images + 0.5
159
160     image_count = 0
161     for row in range(PREVIEW_ROWS):
162         for col in range(PREVIEW_COLS):
163             r = row * (GENERATE_SQUARE+16) + PREVIEW_MARGIN
164             c = col * (GENERATE_SQUARE+16) + PREVIEW_MARGIN
165             image_array[r:r+GENERATE_SQUARE,c:c+GENERATE_SQUARE] = generated_images[image_count] * 2
166             image_count += 1
167
168
169     output_path = os.path.join('/kaggle/working', 'images_created')
170     if not os.path.exists(output_path):
171         os.makedirs(output_path)
172
173     filename = os.path.join(output_path, f"train-{cnt}.png")
174     im = Image.fromarray(image_array)
175     im.save(filename)
176     generator = build_generator(SEED_SIZE, IMAGE_CHANNELS)
177
178     noise = tf.random.normal([1, SEED_SIZE])
179     generated_image = generator(noise, training=False)
180
181     plt.imshow(generated_image[0, :, :, 0])
182     image_shape = (GENERATE_SQUARE, GENERATE_SQUARE, IMAGE_CHANNELS)
183
184     discriminator = build_discriminator(image_shape)
185     decision = discriminator(generated_image)
186     print(decision)
187     # This method returns a helper function to compute cross entropy loss
188     cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
189
190     def discriminator_loss(real_output, fake_output):
191         real_loss = cross_entropy(tf.ones_like(real_output), real_output)
192         fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
193         total_loss = real_loss + fake_loss
194         return total_loss
195
196     def generator_loss(fake_output):
197         return cross_entropy(tf.ones_like(fake_output), fake_output)
198     generator_optimizer = tf.keras.optimizers.Adam(1.5e-4, 0.5)
199     discriminator_optimizer = tf.keras.optimizers.Adam(1.5e-4, 0.5)
200     # This annotation causes the function to be "compiled".

```



```

201 @tf.function
202 def train_step(images):
203     seed = tf.random.normal([BATCH_SIZE, SEED_SIZE])
204
205     with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
206         generated_images = generator(seed, training=True)
207
208         real_output = discriminator(images, training=True)
209         fake_output = discriminator(generated_images, training=True)
210
211         gen_loss = generator_loss(fake_output)
212         disc_loss = discriminator_loss(real_output, fake_output)
213
214
215         gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
216         gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)
217
218         generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
219         discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))
220     return gen_loss, disc_loss
221 general_g_loss=[]
222 general_d_loss=[]
223 def train(dataset, epochs):
224     fixed_seed = np.random.normal(0, 1, (PREVIEW_ROWS * PREVIEW_COLS, SEED_SIZE))
225     start = time.time()
226
227     for epoch in range(epochs):
228         epoch_start = time.time()
229
230         gen_loss_list = []
231         disc_loss_list = []
232
233         for image_batch in dataset:
234             t = train_step(image_batch)
235             gen_loss_list.append(t[0])
236             disc_loss_list.append(t[1])
237
238         g_loss = sum(gen_loss_list) / len(gen_loss_list)
239         d_loss = sum(disc_loss_list) / len(disc_loss_list)
240         general_g_loss.append(g_loss)
241         general_d_loss.append(d_loss)
242
243         epoch_elapsed = time.time()-epoch_start
244         print (f'Epoch {epoch+1}, gen loss={g_loss},disc loss={d_loss}, {hms_string(epoch_elapsed)}')
245         save_images(epoch,fixed_seed)
246
247     elapsed = time.time()-start
248     print (f'Training time: {hms_string(elapsed)}')

```

```

249 train(train_dataset, EPOCHS)
250 generator.save(os.path.join('/kaggle/working', "ak_64_generator_1000epochs.h5"))
251
252
253 plt.plot(general_g_loss, label='Generator Loss')
254 plt.plot(general_d_loss, label='Discriminator Loss')
255 plt.title('Generator and Discriminator Loss')
256 plt.legend()
257 plt.show()
258 plt.savefig(os.path.join('/kaggle/working', 'G_D_loss'))
259
260 plt.plot(general_g_loss, label='Generator Loss')
261 plt.title('Generator')
262 plt.legend()
263 plt.show()
264 plt.savefig(os.path.join('/kaggle/working', 'G_loss'))
265
266 plt.plot(general_d_loss, label='Discriminator Loss')
267 plt.title('Discriminator Loss')
268 plt.legend()
269 plt.show()
270 plt.savefig(os.path.join('/kaggle/working', 'D_loss'))

```

---

## A.4. Generar imágenes

```

1 ak_model = load_model('../input/ak-1000-epochs/ak_64_generator_1000epochs.h5')
2 def generate_N_images(N, model):
3
4
5     output_path = os.path.join('/kaggle/working', 'images_generated_by_ak_model')
6     if not os.path.exists(output_path):
7         os.makedirs(output_path)
8     cnt = 0
9     while cnt < N:
10         cnt += 1
11         seed = np.random.normal(0, 1, (1, SEED_SIZE))
12         image_array = model.predict(seed)
13         filename = os.path.join(output_path, f"img-gen-{cnt}.png")
14         plt.imshow(image_array[0, :, :, 0])
15         plt.savefig(filename)
16 generate_N_images(80, ak_model)

```

---